

MacTech Magazine
Vol. 18, No. 03 • March 2002

MacTech®

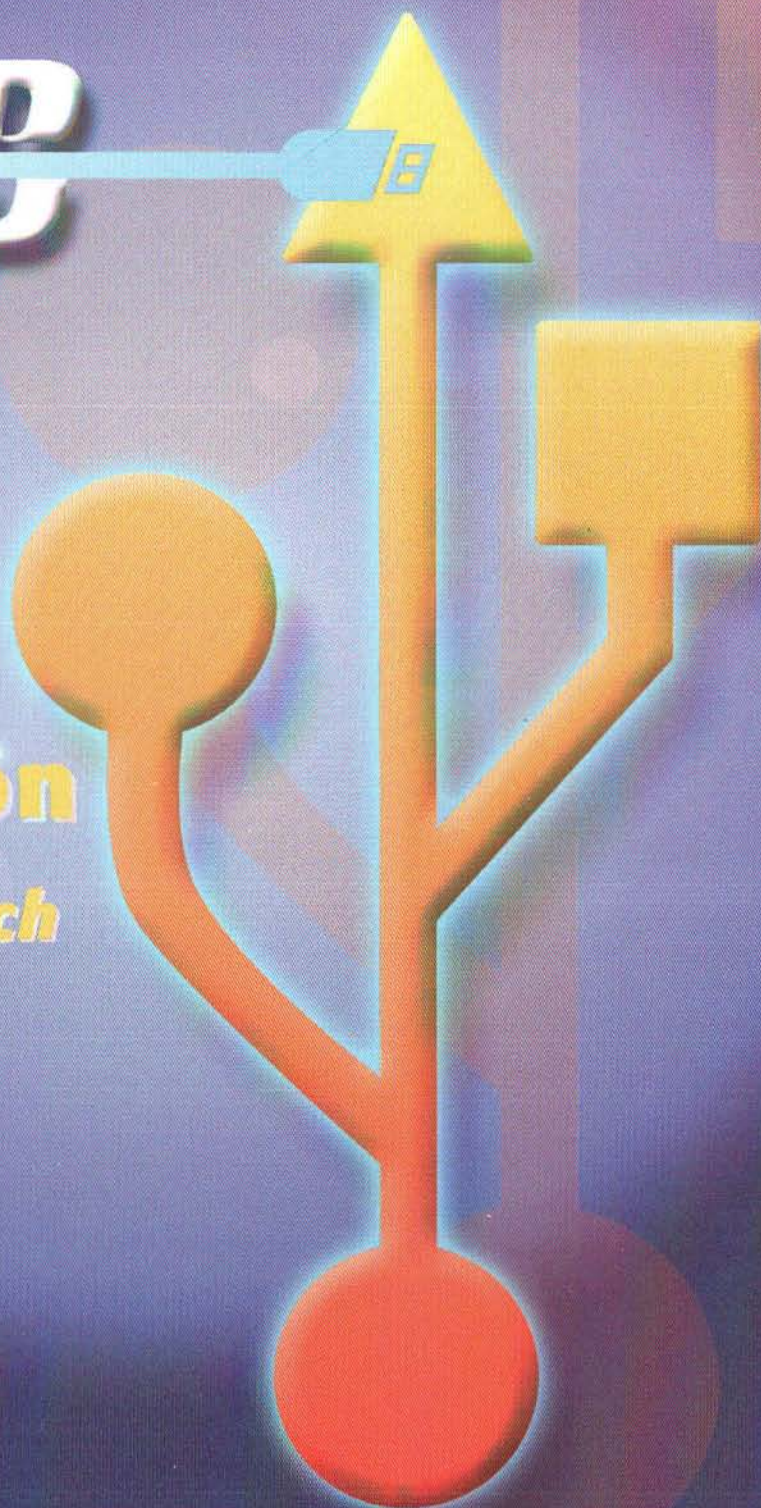
The Journal of Macintosh Technology and Development

1100
USB

USB 2.0 A Comparison

By John C. Welch

\$8.95 US
\$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.



SNIP!



HALF THE CODE*

- **Write once**, run on any platform
- **Database** access & **Internet** protocols
- **Client-server** • **Multimedia**
- **CGI** processing • Rapid **prototyping**
- Blisteringly **fast** • **Easy** to use
- **English-like** programming language

- Develop standalone applications for:
Mac OS 9 | OS X | Windows | Linux

- **Free trial**
- **Email**
- **Phone**

www.runrev.com
info@runrev.com
+44 131 718 4333

"THE HOLY GRAIL OF CROSS-PLATFORM SOFTWARE DEVELOPMENT"

MacUser UK - 5 mice - Nov, 2001

REVOLUTION 1.1

Power to the Developer!



Runtime Revolution Ltd, 91 Hanover Street, Edinburgh, EH2 1DJ, UK

* Based on an independent study comparing Revolution with a leading product. Please see our website for details. Runtime Revolution, the Runtime Revolution logo and Revolution are trademarks of Runtime Revolution Limited, registered in the United Kingdom. All other trademarks are the property of their respective owners.



WHEN THE
SOLUTION
MATTERS

ONLINE SURVEY
FORUMS
CONTACT MANAGER
CALENDAR
TASK MANAGER

P4D PORTAL

REMINDERS

CHAT

AUCTIONS

CLASSIFIEDS

MORE...

Because your new client needs to set project deadlines for a sales team in San Francisco, an IT group in Boston, and a CEO in Barcelona.

Because the Internet is here to stay.

Because deep inside, you've always toyed with the idea of becoming a superhero.

4D Portal, the new, FREE Web application from 4D, Inc., lets you develop customized applications that will change the way people work.

Whether your clients need to solve workflow challenges, streamline communication processes, or brainstorm across time zones, 4D Portal will help you give them what they need, all within a Web browser.

Don't you think it's time you saved the day?

Download 4D Portal free-of-charge today at <http://www.my4DPortal.com/hero>

© 2001 4D S.A. Brands and products referenced herein are the trademarks or registered trademarks of their respective owners.

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail**?

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

DEPARTMENTS

Orders, Circulation, & Customer Service

Press Releases

Ad Sales

Editorial

Programmer's Challenge

Online Support

Accounting

Marketing

General

Web Site (articles, info, URLs and more...)

E-Mail/URL

cust_service@mactech.com

press_releases@mactech.com

ad_sales@mactech.com

editorial@mactech.com

prog_challenge@mactech.com

online@mactech.com

accounting@mactech.com

marketing@mactech.com

info@mactech.com

http://www.mactech.com

The MacTech Editorial Staff

Publisher • Neil Ticktin

Managing Editor • Jessica Stubblefield

Online Editor • Jeff Clites

Regular Columnists

Getting Started – Networking

by John C. Welch

Programmer's Challenge

by Bob Boonstra

MacTech Online

by Jeff Clites

Regular Contributors

Vicki Brown, Andrew Stone,
Tim Monroe, Erick Tejkowski,
Kas Thomas, Will Porter,
Paul E. Sevinç, Tom Djajadiningrat,
and Jordan Dea-Mattson

MacTech's Board of Advisors

Dave Mark, Jordan Dea-Mattson,
Jim Straus, Jon Wiederspan,
and Eric Gundrum

MacTech's Contributing Editors

- Michael Brian Bentley
- Derek J. Burks
- Tantek Çelik, Microsoft Corporation
- Marshall Clow
- John. C. Daub
- Tom Djajadiningrat
- Bill Doerrfeld, Blueworld
- Andrew S. Downs
- Richard V. Ford, Packeteer
- Gordon Garb, Cobalt Networks
- Lorca Hanns, San Francisco State University
- Ilene Hoffman
- Chris Kilbourn, Digital Forest
- Scott Knaster, Microsoft
- Peter N. Lewis, Stairways Software
- Rich Morin
- John O'Fallon, Maxum Development
- Avi Rappoport, Search Tools Consulting
- Ty Shipman, Kagi
- Chuck Shotton, BIAP Systems
- Cal Simone, Main Event Software
- Steve Sisak, Codewell Corporation
- Andrew C. Stone, www.stone.com
- Michael Swan, Neon Software
- Chuck Von Rospach, Plaidworks
- Bill von Hagen
- Eric Zelenka

Xplain Corporation Staff

Chief Executive Officer • Neil Ticktin

President • Andrea J. Sniderman

Controller • Michael Friedman

Production Manager • Jessica Stubblefield

Production/Layout • W2 Graphics

Marketing Manager • Nick DeMello

Events Manager • Susan M. Worley

Network Administrator • Ashutosh A. Gholkar

Accounting • Jan Webber, Marcie Moriarty

Customer Relations • Laura Lane
Susan Pomrantz

Shipping/Receiving • Joel Licardie

Board of Advisors • Steven Geller, Blake Park,
and Alan Carsrud

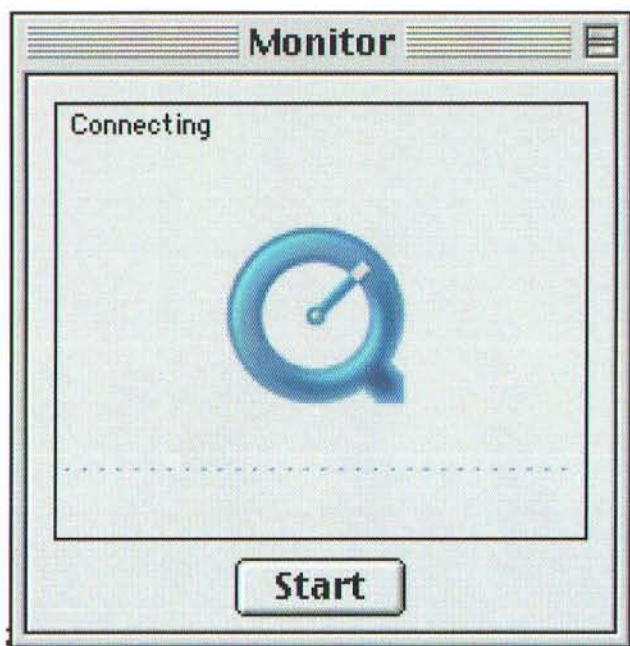
All contents are Copyright 1984-2002 by Xplain Corporation. All rights reserved. MacTech and Developer Depot are registered trademarks of Xplain Corporation. RadGad, Useful Gifts and Gadgets, Xplain, DevDepot, Depot, The Depot, Depot Store, Video Depot, Movie Depot, Palm Depot, Game Depot, Flashlight Depot, Explain It, MacDev-1, THINK Reference, NetProfessional, NetProLive, JavaTech, WebTech, BeTech, LinuxTech, MacTech Central and the MacTutorMan are trademarks or service marks of Xplain Corporation. Sprocket is a registered trademark of eSprocket Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders.

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are \$47.00 per year. Canadian subscriptions are \$59.00 per year. All other international subscriptions are \$97.00 per year. Domestic source code disk subscriptions are \$77 per year. All international disk subscriptions are \$97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

C o n t e n t s

March 2002 • Volume 18, Issue 03



Broadcast News page 44

PROGRAMMER'S CHALLENGE

62 **MegaMinx**
by Bob Boonstra

APPLESCRIPT

27 **AppleScript Studio: Implementing an
Application Preferences System**
by Bill Cheeseman

QUICKTIME TOOLKIT

44 **Broadcast News**
Broadcasting Movies Over a Network
by Tim Monroe

COVER STORY

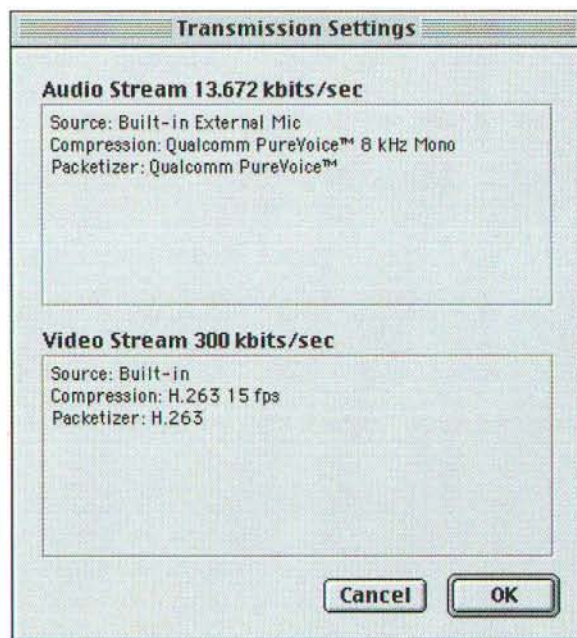
8 **USB 2.0**
A Comparison

By John C. Welch

MAC OS X

4 **What's So Funny About Peace,
Love and Mac OS X?**

by Andrew Stone



QuickTime Toolkit page 44

By Andrew Stone

What's So Funny About Peace, Love and Mac OS X?

One of my favorite riffs is why Apple is so freakin' groovy and it begins something like this: Once upon a time there were two young men in their garage and they dropped some... Well, you know that part of the story - the light shined bright and I think George Harrison was playing. And as Patti Smith sings "This is an era where everyone creates!" And this has come to pass with the help of personal computers, with many of the innovations coming from Apple (and NeXT). The rest of the industry largely copies what Steve Jobs & Co. do! Even the digital hub concept is being ripped off by Intel - they had a full page ad in USA Today the day after Steve introduced the new iMac. MacWorld Expo this year marked the transition into OS X - all new computers are shipped with OS X as the default operating system. One of my favorite parts of the keynote was "a moment of musical tribute" to the late great George Harrison, reminding us of our roots and gratefulness to the visionaries that have come before us. Sales of the new iMacs have been phenomenal, and complementing this hardware renaissance, Cocoa is bound to create a software renaissance.

Running a booth at Expo is both exhausting and exhilarating, but is an essential counterpart to being a software hermit. Meeting your customers, giving demos that really excite people, and making impulse sales are all part of what makes a show invaluable. But what really makes it all worthwhile is the occasional random, ghost-in-the-machine, basic Apple Magic encounters. On Thursday, a rather Tolkien looking fellow ambles up to me and says 'Hey Andrew, here's an invitation to a party we're throwing!' He has an Apple employee badge, so I'm all ears. He hands me a big linen envelope. Thinking of my comrades, I asked for, and received 3 more. "The Gay Blade and the Naked Mole Rats present..." - hmmm, definitely a little edgy, and it starts before the "The Party Formerly Known As ... (The Knife's Last Call). Running in to an old friend who was recently hired by Apple to promote unix ports, we have an impromptu Thai dinner with a bunch of old NeXTSteppers, WebObjects guys,

the original Darwin dude, and a guy from Sun who is looking for people to work on the OS X port of Open Office (www.openoffice.org). Only two guys join me for the parties, but we are rewarded mightily!

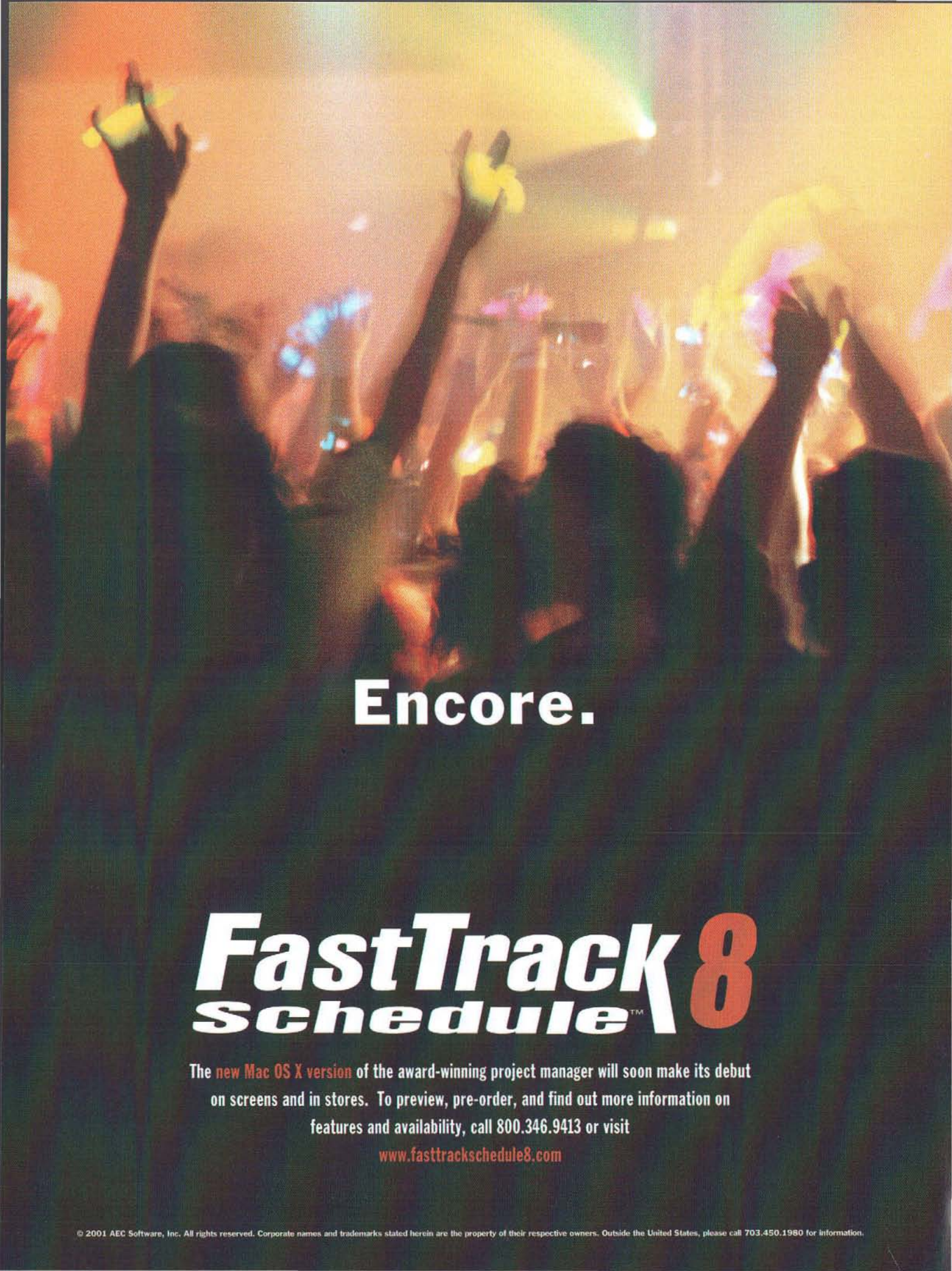
We find the venue in a back-alley south of Market watering hole where there's an open bar with a decent microbrew, and a very famous band (at least in the rock and roll accordion world) takes stage. 'Those Darn Accordions' - <http://thosedarnaccordions.com/index.htm> is a very original and humorous band featuring 3 accordionists, and they totally rocked the house. It turns out I had been invited to a very exclusive Mac insiders party: a veritable who's who in the old time Mac community! During the set break, the band's lead singer, Paul Rogers, turns the audience into the performers. He's a total Mac user, and he breaks out a long list of questions of obscure problems he's had with his Mac, and he wants the audience to answer the questions. One by one, the world's Mac experts crack his questions, and it's a joy to watch our collective knowledge in action.

Finally, he comes to a question - "Mac OS X is on my disk, and its taking up space. What should I do - Nuke it? What's it good for?"

In the spirit of the moment, I jump up on stage, grab the microphone, and say, "I can speak to that. I've been developing with Mac OS X's "new" development environment, Cocoa, for 12 years. I shipped my first app in 1989 on the NeXT, and followed the vision of object oriented software without losing faith, and now have 10 applications shipping for OS X. First - those problems that you've been having on your Mac - most of those (init conflicts, memory issues, crashes) would totally disappear on OS X. Second, we are at the beginning of a renaissance - it was announced at Expo that Apple would be shipping the entire Developer CD with every computer. Who knows where the next killer app will come from?

Cocoa gives tinkerers the ability to write fully integrated OS X software. To give you an idea of Cocoa's power, you can

Andrew Stone <andrew@stone.com> is founder of Stone Design Corp <<http://www.stone.com/>> and divides his time between farming on the earth and in cyberspace.



Encore.

***FastTrack* 8
Schedule™**

The new Mac OS X version of the award-winning project manager will soon make its debut on screens and in stores. To preview, pre-order, and find out more information on features and availability, call 800.346.9413 or visit

www.fasttrackschedule8.com

build a word processor that has multiple fonts, rulers, colors, ligatures, baseline, justification, kerning, full undo, printing, spell-checking and even drag and drop of 30 graphic formats in just 8 lines of code! Putting the development environment into the hands of the users is extremely healthy for the biodiversity and future of OS X software."

Anyway - the second set was even more rocking!

Here's how to create a multi-document word processor that has support for graphics, colors, rulers, alignment, super and subscripting, baseline control, UNICODE, kerning, and it reads and writes RTF and RTFD files. The 10 lines of code are for the reading and writing - the rest is provided via Interface Builder.

1. In Project Builder, select "New Project...", choose "Cocoa Document-based Application", and save it.

```
<00_New_Project.tiff>
```

Project Builder creates the main menu and document interfaces, and the MyDocument subclass of NSDocument.

2. Add the document types that we can edit to the Application Settings pane of the Target Inspector:

```
<01_Document_Types.tiff>
```

Associate the Rich Text Format (RTF) and RTFD (with graphics) file types with "MyDocument".

3. Add the instance variable "text" to MyDocument subclass - only bolded text is what you write, because the rest of the file was stubbed out by Project Builder.

```
//
// MyDocument.h
// XWord
//
// Created by Andrew Stone on Wed Feb 06 2002.
// Copyright (c) 2001 __MyCompanyName__. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@interface MyDocument : NSDocument
{
    id text;
}
@end
```

4. Open MyDocument.nib in InterfaceBuilder, drag MyDocument.h icon from Project Builder to add this instance variable to the File's owner (the MyDocument class).

5. Drag in a text view from the Cocoa-Data Palette window and resize it to fill window

6. In the Size Info, click the "Springs" springy so that it grows to fill the window when you resize it

7. In Attributes Info, click "Undo Allowed" and "Graphics Allowed"

8. Control-drag from the File's owner icon to the text view, and click on "text", and then Connect

9. Open MainMenu.nib, and drag the Text and Font menus from the Cocoa Menus palette onto the main menu. Change occurrences of Newapplication to X Word.

10. Fill out the read and write primitives (the unbolded part of this file was generated by Project Builder):

```
//
// MyDocument.m
// XWord : a 10 line Graphics Enabled Word Processor
//
// Created by Andrew Stone on Wed Feb 06 2002.
// Copyright (c) 2001 __MyCompanyName__. All rights reserved.
//

#import "MyDocument.h"

@implementation MyDocument

- (NSString *)windowNibName
{
    // Override returning the nib file name of the document
    // If you need to use a subclass of NSWindowController or if your document
    // supports multiple NSWindowControllers, you should remove this method and
    // override -makeWindowControllers instead.
    return @"MyDocument";
}

- (void)windowControllerDidLoadNib:(NSWindowController *)
aController
{
    [super windowControllerDidLoadNib:aController];
    // Add any code here that need to be executed once the windowController has
    // loaded the document's window.
}

#define Whole_Range NSMakeRange(0, [[text string] length])

- (NSData *)dataRepresentationOfType:(NSString *)aType
{
    if ([aType isEqualToString:NSRTFDPboardType])
        return [text RTFDFromRange:Whole_Range];
    else if ([aType isEqualToString:NSRTFPboardType])
        return [text RTFFFromRange:Whole_Range];
    // add other types
    return nil;
}

- (BOOL)loadDataRepresentation:(NSData *)data
ofType:(NSString *)aType
{
    if ([aType isEqualToString:NSRTFDPboardType])
        [text replaceCharactersInRange:Whole_Range
withRTFD:data];
    else if ([aType isEqualToString:NSRTFPboardType])
        [text replaceCharactersInRange:Whole_Range
withRTF:data];
    return YES;
}

@end
```

Save the files, build it, test it, ship it! All you need is an icon. Or, you can grab the source from:
http://www.stone.com/dev/X_Word/X_Word.tar.gz

SOFTWARE & DOCUMENT PROTECTION

Have a look at this

Dongles are out!

You are right.
Simple Dongles are obsolete.

WIBU-KEY is in!

WIBU-KEY on the other hand offers additional benefits for your customers on every platform Mac – PC – Unix:

Software goes online

Electronic Software Distribution – safely protected by WIBU-KEY

Web Remote Programming

Transferring data into the WIBU-BOX at the customer's site

Web Authentication

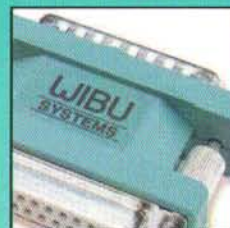
Authentication of the user with the help of his WIBU-BOX

Pay-Per-Use

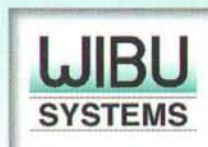
Usage dependent accounting

Mac OS 9 & X

WIBU-KEY supports Mac OS, Windows and heterogeneous networks



www.wibu.com



WIBU-SYSTEMS AG
D-76137 Karlsruhe
WIBU-SYSTEMS USA, Inc.
Seattle, WA 98101
email: info@wibu.com

CeBIT
HANNOVER
13. - 20. 3. 2002
Hall 17 Booth C52



USA, Canada: Griffin Technologies, LLC
phone: (785) 832-2070 · fax: (785) 832-8787
email: sales@griftech.com · www.griftech.com

1-800-986-6578
Order Your Test Kit

Argentina info@safeld-group.com, Australia simone.eckerle@wibu.com, Belgium wibu@impakt.be, Croatia aries@aries.hr, Denmark ivha@danbit.dk,
Finland finebyte@finebyte.com, France info@neol.fr, Hungary info@mrsoft.hu, Japan info@suncarla.co.jp, Jordan, Lebanon starsoft@cyberia.net.lb,
Korea dhkimm@hanmir.com, Netherlands security@unit4.nl, Portugal dubit@dubit.pt, Spain sem@sinix.net, Syria starsoft@cyberia.net.lb,
United Kingdom info@codework.com, USA sales@griftech.com

By John C. Welch

USB 2.0

A Comparison

WELCOME

With all the hype about USB 2.0, I thought it would be a good idea to actually take a look at its protocol, its fundamental parts, and how it works. While USB 2.0 is not everything its backers insist that it is, (and nothing is ever as good as fans would have us believe), it is an improvement on the USB spec, and a way for companies to get higher data rates without having to separately develop for a different standard.

USB 2.0 BACKGROUND

The original impetus for USB was the need for an easily configurable, expandable, fast interface. While the PC and the Mac both had many ways to attach devices, they all had drawbacks. Serial ports were slow, and difficult to configure; parallel ports were limited as to how many devices they could use. PS/2 was a one-device-per-port standard, and it wasn't dynamic; ADB was slow, etc. So Intel, and later others, devised Universal Serial Bus, or USB. USB is an auto-configuring serial bus that allows devices to connect at different speeds (1.5 and 12Mbps) without having to manually assign ids and without needing termination. It is able to (dis)connect without having to shut down the host device. A host device is required (although there is work being done to alleviate this in USB 2.0) so data transfers require host CPU interrupts. The problem with USB 1.X was that after a while, 12Mbps was too slow, especially for things like CD burners, digital cameras, and hard drives. So, USB 2.0 has added a third speed option, 480Mbps.

USB 2.0 also adds support for real-time data transfers, important for video conferencing; support for isochronous transfers, important for devices that need to have guaranteed bandwidth, such as video cameras; and support for mixed mode, or simultaneous asynchronous and isochronous transfers. Because of these new features, USB 2.0 can also support more device types than USB 1.X could.

The benefits for the end user have not changed. You still

have hot-swappable devices, an easy to attach connector, 127 devices per USB channel, automatic driver setup, etc. But as I said, you get isochronous transfers, which are critical for audio and video applications. You also get better support for different packet sizes, error handling, and error recovery.

Like any other bus system, USB has both a physical and a logical topology that it follows. With USB, the physical topology is a tiered, or layered star (think of a star overlaid on a pyramid with specific layers, and that's the general layout.) This topology is limited to seven tiers, with certain tiers having limits as to the type of devices that can be used there. Within this tier, there are two basic types of devices, hubs and functions. Tier 1 is the host, and root hub, and that is the only device at that level. Tier 7 can only have functional devices. You can have a compound device that is both a hub and a function, but those occupy two tiers, with the hub one tier higher than the function. So, you cannot have compound devices at Tier 7. A hub provides attachment points to a USB bus, and a function actually does something, such as a mouse, or a hard drive.

With USB, like SCSI, you have a single host that controls all operations on the bus. Within USB, this is called the host controller, and can be a combination of hardware, software, and/or firmware. All host controllers have a root hub, which provides one or more attachment points, such as the USB ports on the back of your Mac.

USB has both a physical standard for the interfaces on the system, and an electrical standard that provides the signaling characteristics for the bus. In general, USB utilizes a four-wire cable, with two wires used for point-to-point signaling. Three data rates are used in USB:

High, or 480Mbps in USB 2
Full, or 12Mbps
Low, or 1.5Mbps

Note that with USB 2, you can transmit full and low-speed data between the host controller and the hub at the high speed rate, and then between the hub and the end device at either full (high?) or low speed, to minimize the impact of slower devices

John Welch <jwelch@mit.edu> is the IT manager for the MIT Police department. He has over fifteen years of experience at making computers work. He specializes in figuring out ways in which to make the Mac do what nobody thinks it can, showing that the Mac is the superior administrative platform, and teaching others how to use it.

on faster ones. Low-speed support is locked down to a minimal number of devices, such as mice, in order to avoid speed degradation on the bus. In addition to the two data lines on the bus, D- and D+, the other two lines carry power, (Vbus) and the ground, (GND).

USB 2 is a polled bus, with the Host Controller initiating all data transfers. Typically, bus transactions involve three packets, all initiated on a schedule by the Host Controller. The first packet, or 'token packet', describes the type and direction of the transaction, the USB Device address, and the endpoint number. Data transfers from host to device, or from device to host. This information is used by the selected USB device to make sure that it is grabbing data that is intended for it. The source of the data transfer then sends a data packet, or indicates that it has no data. The final packet is then a handshake packet indicating the successful data transmission, or lack thereof. In the case of bus transactions between hubs and host controllers, the transfers involve four packets, and are used to handle data transfers between the host controller and full or low speed devices.

All attachments on a USB bus occur through hubs, which use status bits to report the attachment, or removal of a device to the host when the hub is polled. If the event is an attachment, then the host enables the port, and addresses the USB device via the control pipe at the default address. The host then assigns the device a unique address, and determines if the device is a hub, or a function. The host creates its end of the control pipe using the assigned USB address, and an endpoint of zero. At this time, the USB device also provides the information to completely describe itself, such as the USB standard it is using, the class of the device, and the device vendor. If the attached device was a hub with attached devices, then this happens once for each device on that hub. If the attached device is a function, then the software for that function handles attachment notifications. This is where you will get the notice that you need to go out and get a proper driver for your new USB doohickey.

When a device is detached, the hub disables that port, and sends a removal indication to the host. This is handled on the host by the appropriate software. If the removed device was a hub with attached devices, then the detachment for each of those devices must be handled as well.

The USB 2 spec sets up four basic data transfer types:

- **Control:** Used for device configuration, control, and similar types of data, such as attachment notifications
- **Bulk Data:** For large, bursty data types, with great flexibility in how the data is transmitted. Scanners or printers are examples of this kind of data. Error correction happens in hardware, and the amount of data can vary widely.
- **Interrupt Data:** For timely, but reliable data delivery, where human notice of the data is important, such as the cursor movements from your mouse. This is a limited - latency transfer type, and the speed is determined by the device. Explicit timing is not required, although if the data is interactive, that may pose time limitations.



Fetch 4.0

See Fetch run.



Fetchsoftworks.com

More Features • Fewer Bugs

Bigger Icon

- **Isochronous Data:** For data that needs a specific amount of reserved bandwidth, such as videoconferencing. This data is continuous, and real-time during all phases of data use, so timing related data is critical and explicit. Latency issues are resolved via buffering at the endpoints. This kind of data has less error correction, as things like retries will have a detrimental effect on data timing.

USB COMPONENTS

Host

The first part of any USB system is the host, or the computer. This is the central entity for a USB implementation, and is involved with all aspects of USB functionality, such as bus access and data flow. There is only one host per USB, and each host has three major layers:

- 1) USB Bus interface
- 2) USB System
- 3) Client

The bus interface handles the electrical and protocol layers. This is similar to the interface provided by USB devices, but the bus interface has additional duties imposed on the host of a USB system.

The USB System uses the Host Controller to manage data transfers between the host and USB devices. The actual interface between the USB System and the Host Controller depends on the Host Controller hardware. The USB System works with the host controller to perform translations between the USB clients and the USB interconnect, including various USB features such as protocol wrappers. The USB System is also the layer that handles USB resource management, including bandwidth and bus power, so that clients can access the USB. The USB System is itself made up of three components:

- 1) Host Controller Driver
- 2) USB Driver
- 3) Host Software

The Host Controller Driver, or HCD, maps the various Host Controller implementations into the USB system, so that a client can communicate with its device, without having to know which Host Controller the device is connected to. The USB Driver, or USB D, provides the interface for client to device communications. There is an interface between the HCD and the USB D, known as the Host Controller Driver Interface, HC DI. This is defined by the operating system the Host Controller is running under, and is not directly available to clients.

The USB D also provides methods for data transfer, called I/O Request Packets, (IRP). This is a request to transport data across a specific USB pipe. In addition to this, the USB D gives clients an abstraction of USB devices for configuration and state management purposes. To do this, the USB D owns the default pipe, which is the connection that all USB devices are accessed through for control needs. This pipe is a logical communication between the USB D and the USB device abstraction. If an

operating system provides non-USB software for the loading and configuration of device drivers, then those are used by the drivers, instead of the USB D interface.

The Client layer is the container for the software that directly interact with the USB hardware. This allows the USB system software to manage the client to device communications correctly, preventing the client from directly manipulating the hardware.

Generally, the host, and host layers deal with the following USB functions:

- 1) Attachment and removal of USB devices
- 2) Managing the flow of control data between the host and USB devices
- 3) Managing the flow of non-control data between the host and USB devices
- 4) Status and activity statistic collection
- 5) Managing the electrical interface between the Host Controller and the USB devices, including power management.

There are two ways control information can be passed between a USB device and its host: in-band, or out-of-band. In-band means that the control information is intermixed with the data, whereas out-of-band signaling uses a separate pipe for control information. Every USB device has a pipe that is used for USB standard control flow, called the default pipe. This pipe is only used for standard control information. If the device requires any device-unique information, then it can use the default pipe for this as well. The device can also create additional pipes for specific communications that may not be a part of the standard USB specification. The USB D allows for the sharing of the default pipe with its clients, and provides access to any other control pipes associated with a given device.

As we stated earlier, there are four basic ways to transfer data on a USB: Control, Isochronously, Interrupt, and Bulk. Regardless of the type, all data transfers are managed by the Host Controller. Each device has an interface, or interfaces, that allow a client to communicate with it. Interfaces are made up of pipes that handle the actual data flow between the client and the device, via endpoints on the device. The pipes and interfaces are created by the USB D at the request of the Host Software, which also gives the Host Controller the needed service parameters for configuration requests.

Pipes, which are used for all data transfers on a USB have a number of characteristics based on data rate, is the data steady or sporadic, how long the data can be delayed prior to delivery, and if the loss of the data being transferred will be catastrophic or not. We'll go into more detail on pipes a bit later.

Host Controllers have to manage the various device states that occur on a USB. There are many possible states that a USB device can have, some visible to the USB and the host, some are only visible to the device. The visible states are:

- 1) Attached. The most basic state, this determines if a device is physically on the USB or not. This state does not deal with any other states.

Backup

It's Not as Rough
as It Used to Be...

Automatic Backup System

*The Only Complete Software
and Hardware Solution!*

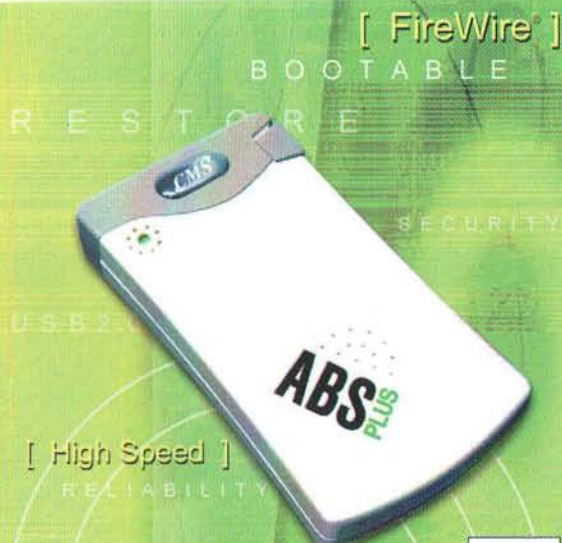
The ABS is The Easiest Way Invented
Backup Your Mac®... Automatically!

Backup, store & share files between iMacs™, iBooks™,
Power Macs™, & PowerBooks™, including video,
digital photography, graphics, & MP3s up to 60GB!

For More Information Call 800 • 327 • 5773 or
lick www.cmsproducts.com/mt1



© 2001 CMS Peripherals, Inc. and the CMS Logo are trademarks of CMS Peripherals, Inc. Mac and the Mac Logo are trademarks of Apple Computer, Inc. registered in the U.S. and other countries. All other trademarks are the property of their respective owners. All rights reserved.



**Now Get 15% Off
All Online Orders!**



- 2) **Powered.** A USB device can obtain power either through the USB, (Bus - powered), or via its own power supply, (Self - powered). One note here, even though a self-powered device can be plugged in and turned on, it is not considered to be in a Powered state until attached to the USB, and Vbus is applied. A device can support multiple powered configurations, depending on the power source. This allows devices to avoid configurations when bus powered that could cause damage to the USB or the host.
- 3) **Default.** This state follows the powered state. Once power is applied, then the device must wait for a reset from the bus before responding to any commands. After the reset is received, then the device is usable. Post-reset, the device speed is determined. The device termination resistors determine if the device is low or full speed is used. If the device is capable of high - speed operation, then the decision as to whether it will operate at high speed is determined by the reset. If a high - speed device is operating in a full - speed environment, then it must reset correctly in that environment, and respond correctly to device and configuration requests, even if it cannot support all intended functionality at full - speed.
- 4) **Address.** Immediately after the reset, the device is using a default address. Once the reset is complete, then the host assigns it a unique address, which is maintained if the device is suspended. Regardless of address, the device responds to any requests on its default pipe
- 5) **Configured.** Once the addressing has been set, the device must be configured. The device receives a SetConfiguration() request, and responds with a non - zero value.
- 6) **Suspended.** If the device has gone a specified period without seeing any bus traffic, then it goes into the Suspended state. In this state, all addressing and configuration data is retained. In addition to this inactivity timeout, the device can also be suspended if the port it is attached to is disabled. Any activity on the device's bus causes it to exit the Suspended state. If the host has entered a suspended state, the device can request the host to wakeup via the use of remote wakeup signaling. This remote wakeup ability must be host configurable.

The overall state of the Host Controller is linked to the states of both the root hub and the USB, as any changes in the Host Controller and/or the attached devices have to be reflected in such a manner that both components stay in consistent states, as in when a resume event occurs following a system waking up from sleep.

Regardless of the function of the end device, all data across the USB is a serial bit stream, that is (de)serialized by the Serial Interface Engine, (SIE) in the Host Controller. To manage the flow of data on the USB, the Host Controller generates frames, (for full speed devices), and microframes, (for high speed devices). To create either, the Host Controller issues a Start Of Frame Token, or SOF. SOFs are issued on a regular basis whenever the Host Controller is in a normal operating state,

every 125µsec for high speed devices and every 1msec for full speed devices. SOFs are generated regardless of bus activity, or lack thereof. If the Host Controller is about to enter a power - reduced state, then it will stop generating SOFs. The SOF token holds the highest priority access to the bus. To ensure that the bus is idle for SOF transmission, the USB hubs contain circuitry to isolate any active transmitters during the End Of (micro)Frame period. To track the current (micro)frame, the host uses two separate methods for full and high speed frames. Full speed frame numbers are at least eleven bits in length, and are incremented every 1msec, the rate of SOF transmission. The lower eleven bits of the frame number id included in each SOF. For high speed devices, the microframe numbers are at least fourteen bits, and increment every 125µsec. Bits three through thirteen are included in each SOF transmission, which means that the same SOF packet value is transmitted for eight consecutive microframes before the SOF packet value increments.

Regardless of speed, when a (micro)frame number is requested from the Host Controller, the number used is the current number at the time the request was fulfilled. The (micro)frame number returned by the host is a 32 - bit number whether the device is full or high speed. At the end of the (micro)frame, there is a quiet period, known as the End Of Frame, or EOF period. During the EOF, the Host Controller ceases transmission, and any transmissions scheduled for the (micro)frame that just passed are retired. If the Host controller was executing a transaction when the EOF starts, that transmission is terminated.

Transmission error handling is the host's responsibility on a USB. While there are many types of transmission errors on a USB, they are all handled in the same manner. Each time an error occurs for a given transmission, an error count is kept, and the transmission is tried again. If the transmission is successful, the error count is reset to zero. If the error count reaches three, then the transmission is retired by the host. In the case of high speed isochronous transfers, there is only one attempt at transmitting data, so no error count is kept.

There are also a number of software mechanisms that are used by the USB host components to allow it to interact efficiently with the USB devices. While there is no real hard division of labor between the host and device interfaces, one requirement for the host is that it supports multiple Host Controller implementations, as defined by the host computer's operating system. The purpose of the host and device interfaces is to allow for different levels of abstraction, while allowing the system to operate together according to the USB spec. Therefore, the HCD provides the abstractions for the Host Controller, and its view of data transfers, and the USB D provides the abstractions for the USB device(s), and their view of USB data transfers. The overall USB system acts as facilitator for these two parts, and their data transfers. Part of this includes things like buffer management and data speed synchronization.

One thing to remember is that when you are talking about software mechanisms, is that they are very dependant on the operating system of the host computer. Linux, Windows, Mac

OS X, and Mac OS 9 all have differences in how they do things, and this extends to USB interactions. In any USB system, on any operating system, there are still basic tasks that need to be fulfilled. Depending on the operating system, the host handles these tasks, in others, the USB system handles them. For any operating system, the USB assumes the existence of a specialized client, called a hub driver, that acts as a central 'clearinghouse' for the addition and removal of devices from a particular hub. When the hub driver receives notification of these events, it uses the host software, other USB clients, and the operating system to recognize and configure the devices.

In the case of device attachment, the hub driver is notified by the physical hub that detects the change. The hub driver uses this hub information to get a device driver from the USB, which sets up the default pipe for the device, and returns a device identifier to the hub driver. From here, there are three configurations that have to happen for the device to be ready for use. First, the Device Configuration. This sets up all the USB parameters for the device, and allocates all the USB host resources that are visible to the device. This is done by setting the configuration value for the device. A limited set of configuration changes/alternate settings are allowed without having to reconfigure the device. Once this step is done, the device is ready, as far as it's concerned.

The next step is the USB Configuration. This is because in order to actually create the USB pipe that is ready for client use, additional information, not visible to the device has to be specified by the client. This is information such as the maximum amount of data the client can transfer per IRP, the maximum service interval for the client, the client notification ID, etc. These values are collectively known as the pipe's Policy, and are used to describe how the client uses the pipe.

The final step is the Function Configuration. Once the first two steps are done, the pipe is ready for use, as far as the USB is concerned. This last step is used for any vendor or class - specific setup the client may need to use the pipe. This is private data between the device and the client, and is not standardized by the USB.

The Host Controller Driver

The HCD itself is an abstraction of the Host Controller data and the Host Controller's view of USB data transmission. The HCDI has the following requirements:

It has to provide abstractions of both the Host Controller hardware, and the data transfers by the Host Controller across the USB, as well as an abstraction for the (de)allocation of Host Controller resources to support guaranteed service to the USB devices. It also has to present the root hub and its behavior in accordance with the hub class definition. This means that the hub driver has to be able to interact with the root hub in the same way as it would for any other hub. Even though the root hub can be implemented in a combination of hardware and software, it has to be able to respond to the device default address, it has to return descriptor

5 times faster

NEW: PRIMEBASE 4.0



Using PrimeBase, you are able to produce Internet and Intranet Applications faster and with the best performance ever.

The development package includes the

- PrimeBase Application Server
- PrimeBase Development Framework
- PrimeBase SQL Database Server

**[get developer key
free of charge]**

download now!
www.primebase.com

Develop on a Mac and deploy without any additional effort on any platform you want.

- Completely cross-platform
- Full-text searching and indexing
- Object-oriented
- Separation of code and layout
- Native connectivity to any database

PRIMEBASE

SNAP Innovation GmbH

Altonaer Poststraße 9a

D-22767 Hamburg / Germany

www.primebase.com

e-mail: info@primebase.com

Fon: ++49 (40) 30629-400

Fax: ++49 (40) 30629-499

information, it has to support having its device address set etc., just like any other hub. The only difference is, with the root hub, since it is closely integrated with the Host Controller, may not require bus transactions for these functions. The HCD is the lowest tier in the USB software stack, and as such, has only one client, the USBD, which maps requests from many clients to the appropriate HCD, which may be managing many Host Controllers. The HCDB is not client accessible.

The USB Driver

The USBD is a collection of mechanisms that are used by operating system components, such as device drivers, to access USB devices. The USBD is the only access to a USB device, and USBD implementations are all operating system - specific. A single instance of the USBD directs access to the HCDs that connect to the Host Controller, and from the client's point of view, the USBD that the client connects to is managing all the USB devices attached to the USB. The USBD presents two types of software mechanisms to clients: command and pipe mechanisms. The command mechanism allow clients to configure and control USBD operations and USB devices, by providing all access to the device's default pipe. The pipe mechanisms allow a USBD client to manage device - specific data and control transfers, but do not allow a client to directly address the device's default pipe.

The specifics of USBD initialization are dependent on the operating system being used. When a USB is initialized, the management information for that USB is also created, and contains the default address device, and the default pipe used to communicate with a newly reset device. When a device is attached to the USB, it responds to a default address until the USB can assign it a unique address, and its default pipe is made available to the hub driver. This occurs during device initialization.

The default pipes are managed by the USBD, and never accessed directly by clients, although they can be used to help fulfill client requests by various command mechanisms. Examples of default pipe data transfers are things like setting a device address. The USBD is directly responsible for allocation and managing appropriate buffering for these transfers. All other pipes are client pipes and are managed by the USBD clients. All buffering and other management of client pipes are handled by the client. There are four types of client pipes, to handle the four types of data transfers we spoke about earlier, namely Isochronous, Interrupt, Bulk, and Control. Along with these four pipe types, there are three basic pipe mechanisms: Abort, for pipe abortion, Pipe Policy Management, and Queuing. These mechanisms in turn support certain basic capabilities, such as Configuration Services, Configuration Management, Initial Device Configuration, Device Configuration Modification Management, Device Removal, Power Control, Event Notification, Status Reporting, Error Recovery, and Remote Wakeup Device Management.

Finally, there are systems, Macs among them, that implement USB services before the operating system loads. This is so you can use Open Firmware with a USB keyboard. In these

cases, as the operating system loads, the Host Controller provides a mechanism that disables this preboot access, and allows the operating system to gain control. Once the operating system gains control, it is responsible for fully configuring the USB. If the operating system has to pass control of the USB back to the preboot system, then the preboot system should treat this as a power up event.

USB Devices

The next USB component to look at are the devices themselves. Each device has three basic parts: The bus interface that handles packet I/O, the middle layer that routes data between the bus interface and the device endpoints, and the top layer that is the device functionality, such as a mouse or keyboard.

When a device is attached, or detached from a USB, the host uses bus enumeration to handle these changes. For example, when a device is attached to a powered port, you get these actions:

- 1) The hub with the new device tells the host via the status change pipe. At this point, the device is Powered, but the port it is attached to is disabled.
- 2) The host queries the hub to determine the precise nature of the change.
- 3) Now that the host has the port for the new device, it waits for at least 100msec to allow the insertion to complete, and the power at the device to be stable. The host then issues a port enable command, followed by a reset command to that port.
- 4) The hub performs the actual port reset. Once this reset signal is released, the port is enabled. The USB device is in the Default state, and can draw no greater than 100mA from Vbus. All of the device's registers and states have been reset, and it answers on the default address.
- 5) The host assigns a unique address to the device, and the device moves to the Address state.
- 6) The host then reads the device descriptor to determine what the maximum data payload that the device's default pipe can handle is, then assigns the device a unique address.
- 7) The host reads the configuration information for all of the device's configurations.
- 8) Based on this information, and the purpose of the device, the host assigns a configuration value to the device. The device is now in the Configured state, and all the endpoints for this configuration are set. The device can now draw needed power from Vbus, and is ready for use.

When a device is detached, the host is notified by the hub. The host then disables the port the device had been attached to, updates its topological information, and releases the device's address.

To support the dynamic environment of the USB, a device can begin interacting with the USB almost as soon as it has returned an ACK following its initial setup, and there are various time limits to facilitate the different stages of device setup. Following a reset, the device has a 10msec interval before

CUT YOUR DATABASE COSTS!

Embedded Hardware

APPLICATION	BENEFIT
Internet Appliances	Small footprint
Medical Devices	Stable
Factory Automation	Includes Full Source Code
Space Exploration	Proven Technology

Vertical Markets

APPLICATION	BENEFIT
Library Management	Easy Deployment
Legal	Flexible Licensing
Insurance	Huge File Support
Inventory Control	Scalable
Point of Sale	Cross Platform



Corporate-Wide Applications

APPLICATION	BENEFIT
B2B Server	Flexible Communication
Proprietary Applications	Fast
Dedicated Web Server	Robust Threading

C-TREE DELIVERS FAST, SOLID, AND FLEXIBLE DATABASE TECHNOLOGY WHILE CUTTING YOUR DEPLOYMENT COSTS...

For over twenty years, FairCom's proven database technology has been a favorite of commercial developers for Macintosh applications ranging from low level embedded appliances to vertical market applications to huge multi-platform, corporate applications. FairCom has established this reputation by delivering uncompromising Standalone and Client/Server technology in a flexible package that enables you to cut costs while utilizing cutting edge technology like our new HUGE file support and 64-bit support. FairCom offers this exceptional performance, unsurpassed data availability and rock solid reliability with a low total cost of ownership and flexible licensing options.

Get more for your development dollar! Visit www.faircom.com/ep/freecdoffer today for a free developer's CD.



FairCom
CORPORATION

FairCom

Offices

USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

Mac Support Since 1985 • www.faircom.com • USA. 800.234.8180 • info@faircom.com

Other company and product names are registered trademarks or trademarks of their respective owners.

© 2002 FairCom Corporation

it is expected to start processing requests. This is considered a recovery period, also applicable to resuming operation following a suspend. After a recovery interval, of the device receives a request to set its address, it has to be able to complete this request within 50msec. Once the device has processed the set address command, it sends an acknowledgement back to the USB indicating that the device address was successfully set. At this point, there is another recovery interval of 2msec, following which the device must be able to respond to Setup packets sent to the new address.

If the device handles device requests that need a data stage transfer to the host, then the device must be able to return the first data packet to the host within 500msec of the first request. All subsequent packets must be returned within 500msec of the successful transmission of the previous packet. After returning the final data packet, the device must finish the status stage within 50msec of returning the final data packet.

If the device is handling device requests that need a data stage transfer to the device, then the USB has an upper limit of 5 seconds to process a command, although this is not a hard limit, and can change depending on need.

Since a high - speed device has to be able to operate at both high - and full - speeds, it has to be set for the speed of the USB, depending on bus speed limits. This is normally done as part of the reset processing. Once the device speed has been set, it doesn't change until the next time it is reset. The high - speed device must report to the USB any limitations in its functionality that may exist for a given speed. It does this during configuration by returning descriptors for its current operating speed. If need be, the USB can get descriptors for both modes, and set the device to the USB's limitation, or device use on that USB.

All USB devices respond to host requests on their default pipe via control transfers sent in a Setup packet, which is eight bytes in length. Each Setup packet has five fields that define the request, and its parameters. The first field is `bmRequestType`, and identifies the specific request parameters, such as direction of data transfer, the type of transfer, and the recipient of the transfer. `bmRequestType` is one byte long. The next field, `bRequest` specifies the particular request, is one byte long, and is modified by the type information in `bmRequestType`. `wValue`, the third field, is two bytes long, and is used to pass a request - specific parameter to the device. The next field, `wIndex`, is similar in function to `wValue`, and is also two bytes long. The final field, `wLength`, is also two bytes long, and shows the length of the data that will be transferred during the second phase of the control transfer. Note that the data transferred may have a length that is less than `wLength`, but never greater.

As we saw earlier, USB devices respond to configuration information requests via descriptors. A descriptor is simply a way of storing device capability descriptions in an easy - to - retrieve manner, that can reuse parts of other configuration descriptions for that device, much like a database record can use similar information that may be contained in other records or fields. Each descriptor starts with a one byte

field that contains the total number of bytes in the descriptor, followed by a one byte descriptor identifier. If appropriate, the descriptor can contain human - readable information in the form of strings. If the descriptor length value is less than the specification - defined length for that descriptor, then it is rejected by the host. If the descriptor is larger than the defined length, then the extra data is ignored.

If the descriptors are class or vendor - specific, then there are two ways for the device to deal with them. If they use the standard descriptor format, then they are returned as a standard descriptor would be returned, and can be interleaved with standard descriptors. If they modify a standard descriptor, then they must follow that descriptor. If they use a non-standard format, then the descriptor request specifies the class/vendor - specific descriptor and index, to retrieve that descriptor from the device.

There are seven standard device descriptors, which can only be modified by modifying the USB specification, and are:

- 1) Device. This descriptor gives general, global, non-configuration - specific information about the device. If the device is a high - speed device, and has different information for high and full - speed operations, it must also have a `device_qualifier` descriptor. If the device is capable of high-speed operation, then it returns a 2.0 as its version number. If the device is a full or low - speed only device, then it returns an error to a request for the `device_qualifier` descriptor.
- 2) `Device_qualifier`. This descriptor contains the information about a high - speed device that will change if the device is operating at a different speed.
- 3) Configuration. The configuration descriptor contains a specific configuration's information. So if a USB camera has different operational characteristics depending on the camera's settings, each of these gets its own configuration descriptor.
- 4) Interface. This describes a specific interface within a given configuration, and a given configuration can have multiple interfaces, each with its own descriptor. The interface descriptor is always returned as part of a configuration descriptor, and cannot be accessed separately.
- 5) Endpoint. This descriptor is for each endpoint within a given interface. Like the interface descriptor, it is returned as part of the configuration descriptor, and cannot be accessed separately.
- 6) String. These descriptors are optional, and are UNICODE strings. They are used if there is a need for human - readable descriptors.

USB Hubs

The third physical component of the USB spec is the hub. USB hubs are a critical part of the USB, and do much of the work involved in making USB as easy to use as it is. Hubs must

Application Builder Collection™

Reusable Cocoa frameworks for building solid Mac OS X applications

- Port to Cocoa faster
- Save thousands by reducing coding and testing time
- Spend your time developing the features that make your software stand out
- Quickly create professional looking applications
- Utilize object-oriented technology for easy customization and reuse
- No deployment licenses or royalties!

Collection Includes:

- Rotational Slider
- Formatters for Currency, Phone, Text, and Custom Fields
- Draggable Cocoa Cells for Tables
- Calendar View and Popup
- URL Clickable Field
- Twiddle View
- Archive Builder Application
- And More . . .

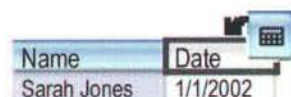
Learn more at www.jiiva.com



rotate images and more



control data input and presentation



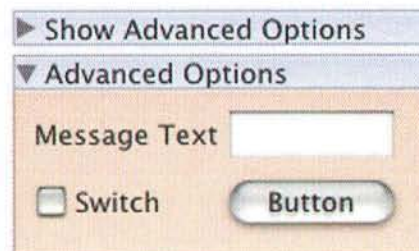
drag sliders and other widgets into tables



add calendar popups and views

Visit Jiiva's [web site](http://www.jiiva.com/web_site)

turn text fields into clickable URLs



create complex interfaces with versatile twiddles



 **Jiiva**, Inc.
www.jiiva.com/abc

deal with connection behavior, power management, device (dis)connection management, fault detection and recover, and support for high/full/low - speed devices. Hubs in a high - speed environment must also isolate the high - speed signaling from the full/low - speed signaling to ensure that both interoperate correctly.

There are three basic components to the USB hub. The first is the Hub Repeater, which manages connectivity between same - speed upstream and downstream ports. The repeater also handles the hub entering and leaving the suspend state, including remote wakeups. The next component is the Hub Controller which manages host access to the hub along with providing status and control. Finally the Transaction Translator handles split transactions, (Split transactions are, for example, a full - or low - speed transaction on a high speed hub that has low or full speed devices attached.) The speed of a device on a downstream port determines whether the port is connected to the Hub Repeater, or the Transaction Translator.

If the upstream port on the hub is attached to a full or low speed environment, any high - speed functionality is disabled. This also disables the transaction translator, and all ports are connected to the hub repeater. If the upstream port is attached to a high - speed environment, then the full/low - speed repeater is disabled, and the hub repeater is a high - speed repeater. If a full/low - speed device is attached, then the port must connect to the transaction translator.

For data traveling upstream, the connection is a line, in that one downstream port connects to the upstream port of the hub. If there are many hubs in line to the host, then each hub only connects a single downstream to upstream port. If the data is flowing downstream, then the transmission is more of a broadcast, whereby all active downstream ports on that hub get the data from the upstream port, regardless of what device the data is actually intended for. If the device isn't supposed to get the packet, then it discards the packet.

USB Bus Electro/Mechanical Characteristics.

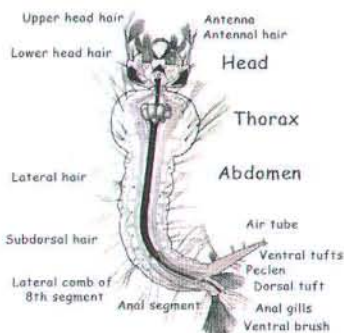
The next thing to look at is the actual USB physical bus characteristics. USB 2's different operating modes have different mechanical and electrical requirements, and there is a nominal dividing line, with low speed, (1.5Mbps) on one side, and High/Full speed, (480/12Mbps) on the other. For example, High and Full speed both require a shielded cable with twisted pair wiring, whereas that cable is only recommended for low speed operation.

USB uses a keyed connector protocol, both for ease of use, and to minimize termination issues. There are two types of connectors, Series "A" and Series "B". Series "A" connectors are the familiar flat connectors that you see on the back of modern Macs, and are used for upstream, (towards the host system or hub), connections. Series "B" connectors are the smaller, square - looking connectors and are normally only used for downstream, (towards the device) connections. In any case, a Series "A" connector can only attach to a Series "A" receptacle, and the same holds true for Series "B". (you may be asking about devices that have a Series "A" receptacle requiring a cable with two Series "A" connectors. Remember that a device can also be a hub, and that hub connectors are Series "A" connectors. So things like keyboards that can be hubs and devices use Series "A" connectors, whereas a printer that is only a device would use the Series "B" connectors.)

USB cable consists of four conductors. Two are used for power, VBUS and GND, the other two are used for data, D+ and D-. Within the cable, the conductors are color coded as follows: Red=VBUS, Black=GND, Green=D+, White=D-. The pinout of USB connectors is standardized as well: 1=VBUS, 2=D-, 3=D+, 4=GND, and the cable shell acts as a shield and drain wire. There are three types of cables in the USB spec, standard detachable cable, high/full - speed captive cable, and low - speed captive cable. (In plain English, a captive cable is hardwired to the device, ala an Apple keyboard.)

Standard detachable cable has a Series "A" connector on the hub end and a Series "B" connector on the device end. These cables should only be used for high/full - speed devices, as they can exceed the length limitations for low speed devices. This cable must also meet impedance, signal pair propagation and delay requirements for high/full - speed devices. The GND conductor doesn't actually earth the device or hub, but provides a common ground reference between the up and downstream ports. The maximum cable length is limited by the voltage drop

Study bugs?



Or **KILL** them?

QC

New Version

Upgrade Now \$29

www.onyxttech.com

*Detect stale handles, runtime block overwrites,
DisposeHandle on resources, invalid BlockMoves,
writes to location zero, Validate Handle/Pointer*

across the GND conductor.

High/full - speed captive cable is identical to standard detachable cable, except that one end is hardwired to the device, and that it can be used with low - speed devices as well as high/full - speed devices. If it is used with a low speed device, then the cable must meet all low - speed requirements.

Low - speed captive cable is identical in design to high/full - speed captive cable, but may only be used with low - speed devices. Any other cable designs are prohibited by the USB spec.

USB Electrical Characteristics

In addition to the bus electrical characteristics we just looked at, there are quite a few other electrical characteristics in USB, such as power distribution, signaling, and physical layer specifications. With the addition of high - speed mode in USB 2, there are some additional requirements that affect the electrical characteristics of USB. Things like the requirement that all USB 2 - compliant hubs support high - speed mode, any downstream transceiver must support all three modes, and that the cabling used support all three modes are examples of this.

The new 480Mbps data rates in USB 2 require new signaling and termination characteristics to enable reliable operation at these speeds. High speed cable is terminated on both ends with a resistance from each wire to ground. This resistance is set to one - half the overall differential impedance of the cable, which is 45 ohms, for a total differential termination of 90 ohms

When initiating a high - speed transmission, the transceiver starts an internal current source, derived from the positive supply voltage for the transceiver, and sends that current down either D+ or D- via a high - speed current switch. If the current is sent down the D+ line, then a J state is being signaled, and if the D- line is used, then a K state is signaled. Data encoding follows an NZRI scheme (NZRI is an encoding scheme where each time a '0' is sent, the value of the signal changes from low to high, and vice-versa. USB 2 uses NZRI with bit stuffing, which adds a '0' after any six consecutive '1's, to force a transitions, and aid with clock synchronization.) The actual voltage levels are determined by the magnitude of the current source, and the value of the termination resistors. As stated before, the resistance from either D+ or D- to ground is 45 ohms, and the output voltage differential between lines is plus or minus 400mV at that resistance. The differential voltage is used to detect things like a bad connection, or that the device has been disconnected.

As far as actual signal levels go, in general, the difference between a '1' and a '0' is 200mV for differential systems. When in a J state, low speed connections are signaling a differential 1, and full speed connections are signaling a differential 0. For the K state, these are reversed.

In low/full - speed systems, the SOP is signaled at the origination port by driving both D+ and D- from the idle state to the K state. To signal the End of Packet, EOP, D+ and D- are driven to a J state, then to the Idle state, and the bus termination hold the bus in that state. In a high speed connection, the Idle state is when both D+ and D- are nominally at GND. The SOP

is signaled by driving both data lines into a K state. The EOP and idle state are achieved by simply halting current flow into the data lines, and returning them to GND.

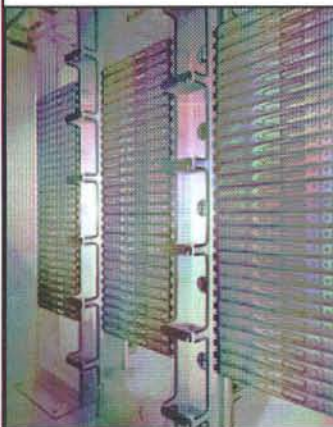
Another function of the USB electrical system is that of power distribution. As with ADB, and PS/2 even, the bus supplies power for devices to operate. Unlike the other two though, you can also have a self-powered device that doesn't need bus power, an important consideration when you are trying to hook up over a hundred devices on a single bus. Finally, there are also facilities to warn that a device needs too much power, and disable that device, without shutting down the whole bus, or worse yet, burning out any components.

To facilitate easier operation of the power distribution system, USB introduces the concept of a unit load. A unit load is a 100mA load on the bus. A device that only draws 100mA is said to use one unit load, and is a low power device. A device can draw up to 500mA, or five unit loads, and is then considered a high power device. All devices default to low-power, but can transition to high power, although that is under software control. The software has to make sure that going to high power will not damage the bus, *before* allowing that transition to happen, thereby avoiding some of the problems that an overloaded ADB bus could cause.

There are a number of both power source, and power consumption devices, including:

Webmasters! Start your web hosting business with edition.net reseller hosting!

- dedicated platforms
- single and multi-domain hosting
- your choice of Mac OS or Linux platforms
- FileMaker Pro hosting featuring Lasso
- QuickTime streaming server
- Our web-based control panels make it easy to
- deploy your multi-domain server



**edition.net –
the Fine Arts
Network**

<http://www.edition.net> • info@edition.net
+1 (877) 225-3821 toll free • +1 (562) 430-5953 international

- Root port hubs. These attach directly to the USB Host Controller, and derive power from the same source as the host controller. If the hub is powered via AC or DC external power, then it must supply at least five unit loads to each port. If the hub is battery powered, then it can supply either one or five unit loads. A five unit load port is a high power port, and a one unit load port is a low power port.
- Bus - powered hubs. These draw all power from Vbus on the hub's upstream port. They can only draw up to one unit load on power up, and up to five following configuration. This is split amongst the hub, its ports, and any other hub functions. The ports on this hub are low power ports, and must maintain that current when the hub is in the Active, or the Suspend states. This type of hub can have five ports, one upstream, and four downstream. If more downstream ports are required, then the hub must be self-powered.
- Self - powered hubs. These do not use Vbus as a power source. This hub may draw a unit load from Vbus to keep the interface working if the rest of the hub is powered down. As with the root port hubs, if a self - powered hub is supplied by external AC or DC power, then all ports must be high power ports, if the hub is battery powered, then the ports may be high or low power ports.
- Low - power bus functions. All power comes from Vbus, and they can draw no more than one unit load.
- High - power functions. Uses Vbus as its power source. Draws up to one unit load on power up, and up to five unit

loads following configuration.

- Self - powered function. No operating power from Vbus, but may use one unit load from Vbus to keep the interface working when the function is powered down.

To avoid problems in wrong - way power distribution, no device supplies power to Vbus on the upstream facing port.

USB Bus Protocol

Like any other networking protocol, USB has a protocol layer that defines packet layout, media access, etc. USB is a little-endian protocol, in that it sends data across the line starting with the least significant bit, and ending with the most significant bit. (Considering that USB was started by Intel, the largest supplier of little-endian CPUs, this is not surprising.) Byte - ordering is more important to lower layer functions than user - level applications, as the upper layers of the OSI model handle byte - order issues.

All packets begin with a SYNC field that is used to align the data with the local clock. The size of the SYNC field depends on the speed of the USB, with 8 bits being used for low/full speed and 32 bits used for high speed. This field is only used for synchronization. The last two bits of the field are used to identify the end of the SYNC field. The Start of Packet delimiter is also part of the SYNC field.

Take



With JobOrder

Estimating ■ Scheduling ■ Job Costing ■ Traffic ■ Billing
Reporting ■ Notifications ■ Accounting ■ Calendars

Discover the Affordable Management Software

free demo at www.JobOrder.com

toll-free 877.714.2587 phone 607.756.4150

JobOrder™
maximizing business productivity

Introducing

**Collaborative Commerce
for Small-to-Midsize Businesses
and for People, too**

**For
Mac OS X
Linux
Windows**

MyBooks®
MyBooks Professional
Appgen® Custom Suite
Moneydance®

Fourth generation
development environment

Variable-length
relational database

Commerce modules

Source code

visit www.appgen.com or
call 1 800 231 0062



APPGEN®
Collaborative Commerce Platform

Immediately after the SYNC field is the Packet Identifier Field, or PID. The PID is made up of a four - bit packet type, and a four - bit check field. The type field indicates the type and format of the packet, and the check field assists in the reliability of decoding the packet. Each bit in the check field is a complement of it's respective bit in the type field.

Addressing is done via two fields, function address and endpoint. Functions must always decode both fields, and a mismatch on either causes the packet to be ignored. Aliasing of either field is not permitted. The function address field specifies either the source or the destination of the data packet. The endpoint field is used when more than one endpoint per function is required.

The frame number field is eleven bits in size, and is incremented by the host on a per - frame basis. The maximum frame number is 7FFH, and the counter rolls over upon hitting this.

The data field can be anywhere from zero to 1024 bytes in size. Data within each byte is shifted out LSB first.

USB 2.0 uses two CRC checks, one for the token, and a second separate one for the data bits. The token CRC is five bits, the data CRC is sixteen bits. The token CRC does not cover the PID, which has its own check field.

USB Data Flow

Our final section deals with the data flow within USB 2.0. This is where all the various sections of the bus work together

to get your device to operate cleanly with your Mac, PC, whatever.

The first aspect we have to look at here is the endpoint. The formal definition of a USB endpoint is that it is "a uniquely identifiable portion of a USB device that is the terminus of a communication flow between the host and device." Remember that each USB device can have multiple endpoints, all grouped together in a logical device. This logical device is what is given an address by the USB when it is attached and configured. Within the logical device, each endpoint has it's own unique identifier that is determined by the device. Each endpoint also has its direction of data flow determined by the device. The combination of logical device number, endpoint number, and directional flow ensures that every endpoint is able to be uniquely referenced. (This is logical, as you wouldn't want data for your USB printer sent to your keyboard.) Because all endpoints are simplex, the direction of data flow is able to be used to help reference that endpoint. (More simply put: Endpoints are either gozintas or gozoutas, not both.)

In addition to the other identifiers, the endpoint has other characteristics that help describe it and its capabilities, such as bus access frequency requirements, bandwidth requirements, error handling requirements, maximum packets size, and transfer types. All endpoints other than endpoint zero are in an unknown state until after device configuration, and cannot be handled until configuration happens.

Endpoint zero is used to implement a default control method to and from the device. The USB system software uses endpoint zero to initialize and manipulate the device as the default control pipe, which supplies access to the device configuration information, and allows for generic status and control access. The default control pipe supports control transfers, and is available once the device is attached, powered, and has received a bus reset.

Functions can have additional endpoints as needed, limited by the device speed. Low speed functions can only have two additional endpoints, whereas full speed endpoints can have an additional 30 endpoints, (15 input, 15 output.) These endpoints are unavailable until the device has finished configuration.

The next structure is the Pipe, which is the association between the endpoint on a device, and the host software. Pipes are how data moves between memory buffers on the host, and the device endpoints. There are two types of pipe communications, each mutually exclusive: Stream, or unstructured data flow through the pipe, and Message, or structured data flow through the pipe. Regardless of structure, or lack thereof, the USB doesn't interpret the data moving through the pipe(s).

In addition to data structure, pipes have some other characteristics, such as bus access and bandwidth usage, transfer types, and the characteristics of the endpoint the pipe is connected to. If the pipe is connecting two endpoint numbers zero, then it is known as the Default Control Pipe. This pipe is available after the bus reset, but prior to configuration, whereas

<http://www.scientific.com>

Professional Software Developers

Looking for career opportunities?

Check out our website!

Nationwide Service

Employment Assistance

Resume Help

Marketability Assessment

Never a fee

Scientific Placement, Inc.

800-231-5920 800-757-9003 (Fax)

das@scientific.com

other pipes are unavailable until after configuration. This is because the default control pipe is used to configure the device, and to determine the device identification. Once this is done, then the default control pipe can be used by the device specific software. However, the USB system software owns the default control pipe, and mediates its usage by other client software.

As we said before, there are both stream and message pipes in a USB. Stream pipes deliver unstructured data between endpoints. Data flows in one end and out of the other, and stream pipes are always unidirectional. Stream pipes interact with what the USB sees as a single client. If multiple clients are using a single stream pipe, then no synchronization is provided for those clients by the USB. Data flow in a stream pipe is sequential, in a FIFO method. Since stream pipes are unidirectional, they are bound to a single endpoint that is handling data in the appropriate direction. Stream pipes support bulk, isochronous, and interrupt transfer types.

As we said before, message pipes pass structured data. There is also a different method towards how the data is transferred. Instead of just jamming data down the pipe, a request is sent to the device from the host. Then the data transfer happens in the appropriate direction, followed by a status stage. This data flow procedure is the reason for the structuring of the data in the pipe, so that the different data types can be identified. Message pipes can be bidirectional, and the default control pipe is always a message pipe.

Since the device is only required to service a single message request at a time per message pipe, the USB system software ensures that multiple concurrent requests are not sent to the pipe. Multiple software clients on the host can make requests via the default control pipe, but they are sent to the device on a first - come, first served basis. Data flow during the data and status stages is determined by the device. A message pipe to a device request a single device endpoint number in both directions, and are used for the control transfer type.

We have previously stated that there are four data transfer types in a USB: Control, Isochronous, Interrupt, and Bulk. Let's take a look at these in more detail.

Control transfers allow access to different parts of a device. They are designed to support configuration - type commands, such as configuration and status types of communication between client software and the function it needs to access. These communications, done via message pipes, again, use the three part request/data/status model for data flow.

All USB devices have at least one control pipe, the default control pipe, but can have more if need be. The packets that travel via control transfers have four possible sizes, depending on the speed of the transfer: 8 bytes for low - speed transfers, 8, 16, 32, or 64 bytes for full - speed transfers, and 64 bytes for high - speed transfers. This size applies to the payloads of data packets, and is signaled by the endpoint via the device descriptor. Status packets are always 8 bytes.

Although control transfers can run as fast as the USB will let them, there is no way to guarantee a specific speed

for a control transfer. Although the USB requires that part of each (micro)frame be available for control transfer usage, they are not allowed to take needed bandwidth from data transfers. If need be, the USB system software can also alter the speed of control transfers. As well, the total number of control transfer bandwidth usage is limited, to avoid control transfers stealing too much bandwidth.

The next transfer type in USB 2.0 is Isochronous, familiar to Mac users via FireWire. In USB 2.0, a request for an isochronous transfers gains the requester three features of that transfer, namely guaranteed access to USB bandwidth, including latency bounding, guaranteed data rats through the pipe as long as there is data to deliver, and no retrying of data delivery in case of error. (This may sound odd at first, but error recovery takes time, extra packets and bandwidth. This means that during error recovery, keeping the guaranteed bandwidth available is difficult, so error recovery is ignored.)

Isochronous data is transferred via a stream pipe, with no structure imposed on it by the USB. Since this is a stream pipe transfer, isochronous data flows are unidirectional as well. Maximum data payload sizes are established by the endpoints of an isochronous pipe within the limits of the bus speed for that pipe, (1023 bytes for full - speed, and 1024 bytes for high - speed.) During configuration, the USB system software tries to ensure there is sufficient bandwidth for the maximum data size of the transfer. If there is, only then is the configuration established.

Felt Tip Software presents

Sound Studio 2.0

Built for Mac OS X

With Sound Studio 2.0, you can:

Digitize cassette tapes and vinyl records to make CDs

Record live performances and interviews

Record your favorite radio programs with Timer Recording

Edit out dead air, glitches, and mistakes in recording

Apply any of 19 effects including Reverb and Chorus

Create new sounds and loops

Sound Studio is an audio recording and editing application built for Mac OS X. It can also run on Mac OS versions 8.1 and up. Some systems may need a USB audio input device to record.

Only \$49.99

Download 14-day trial version at
<http://www.felttip.com/ss/>



felt tip software

807 Keely Place, Philadelphia, PA 19128, USA
www.felttip.com

Isochronous transfers are limited to full/high - speed USB implementations, and USB limits the percentage of a (micro)frame that can be used for isochronous transfers at 90% for full - speed transfers, and 80% for high - speed transfers. In general, hosts must not issue more than one transaction per (micro)frame, unless the endpoint is high - speed and high - bandwidth, which allows for multiple transactions per (micro)frame, and a maximum speed of 192Mbps. Due to limitations in the USB bus frequency and (micro)frame timing, no more than 151 full - speed one byte data payloads per frame, and no more than 193 one byte payloads per microframe for high - speed transfers.

To allow for guaranteed bandwidth in isochronous transfers, there is no error retransmission on the bus. This means that things like handshakes are not used either. Basically, it is assumed that all isochronous data transfers succeed.

The third type of data transfer is the Interrupt transfer. This is used for devices that don't need to send data often, but require bounded service periods for their data transfers. Characteristics of an interrupt transfer are a guaranteed maximum service period for the pipe, and a retry of transfer attempts for failed delivery during the next service period. Like isochronous transfers, interrupt transfers use stream pipes, and have no structure imposed on them by the USB. They are also unidirectional.

Data payload size limitations for interrupt transfers are 8 bytes for low speed transfers, 64 bytes for full - speed transfers, and 1024 bytes for high - speed transfers. As well, high - speed transfers can request two or three transactions per microframe. The maximum size in an interrupt transfer is determined during device configuration, and remains constant for the life of that configuration. If there is sufficient bus time for that data payload size in the assigned period, then the configuration is established.

Frame usage maximums for interrupt transfers are limited to 80% for high - speed and 90% for full/low - speed transfers. Per - frame one byte payloads are limited as follows: low: 10, full: 108, high: 134. Maximum data speed for high speed endpoints in an interrupt transfer is 192Mbps.

Bus access periods for interrupt transfers are determined by the speed of the endpoint. Low speed endpoint periods range from 10 to 255msec, full - speed transfers need from 1 to 255msec, and high - speed period times are based on the equation $(2^{bInterval}-1) \times 125 \mu s$, where $bInterval$ is a value from 1 to 16. The periods between access times will be no longer than determined by the configuration, but can be shorter, depending on bus needs. As well, the access period is only used to check for IRP packets from the software client. If the access time for an endpoint comes around, and there is no IRP packet, then the endpoint is not polled. This avoids the overhead of polling endpoints with no data transfer needs. Like isochronous transfers, the maximum data rate for a high - speed endpoint is 192Mbps.

The fourth transfer type is the Bulk transfer. This is for devices that need to move large amounts of data, but at irregular

intervals, and need to be able to get as much bandwidth as possible. Characteristics of a bulk transfer include USB access on a bandwidth available basis, error retransmission, and guaranteed delivery of data, but no bandwidth or latency guarantees. Since the transfers are on a bandwidth available basis, the speed of the transfers is dependent on available bandwidth. As with Isochronous and interrupt transfers, bulk transfers use stream pipes, are unidirectional, and have no USB - imposed structure.

Packet sizes for bulk transfers are 8, 16, 32, or 64 bytes for full speed endpoints, and 512 bytes for high - speed endpoints. Low speed endpoints do not use bulk transfers. If a data payload is not exactly one of the usable sizes for the endpoint, there is no requirement for padding the data out to those sizes. This also applies to isochronous and interrupt transfers.

When an endpoint needs to use a bulk transfer, it cannot 'reserve' bandwidth, ala isochronous, nor can it reserve an access time, ala interrupt. The USB notes the size of the transfer, the speed of the endpoint, and makes the best effort it can to move the bulk transfer data as fast as possible given the conditions on a specific bus. Essentially, bulk transfers get what is left over from interrupt and/or isochronous transfers. The 'fairness' of this policy is determined by the Host Controller. Due to the way bulk transfers work, client software cannot, and should not assume any specific speeds or access times for this transfer type, nor should it assume any particular order with regard to control transfers.

Bus frequency and frame timing limit the data payloads for bulk transfers to less than 72 full - speed eight byte payloads and less than 14 high - speed 512 byte payloads. Regardless of payload size, bulk transfers do not guarantee that the payload will be delivered in any number of (micro)frames. Due to USB conditions, a transfer could be spread out across multiple (micro)frames, even if it is small enough to fit in a single (micro)frame. As with isochronous and interrupt transfers, maximum speed for a high - speed endpoint is 192Mbps. High - speed endpoints can have between one and three transactions per microframe to support high - bandwidth endpoints.

Regardless of the type of transfer, there is a great deal of management that occurs to ensure that the USB functions smoothly, and efficiently. To handle this task, there are a number of USB entities that shared the load of transfer management, mainly the client software, the USBDB, and the HCD.

One of the critical tasks in bus management is obviously transfer management. To ensure that all transfers are serviced correctly requires that the various USB entities work in a coordinated fashion. The first of these entities is the client software, which determines what transfers need to be made with a given function. It makes use of the appropriate operating system interfaces to request IRPs, ensures the proper pipes are used for the various transfer types, and helps manage the requirements of the different transfer types. All requests made by client software are presented via the USBDB interface.

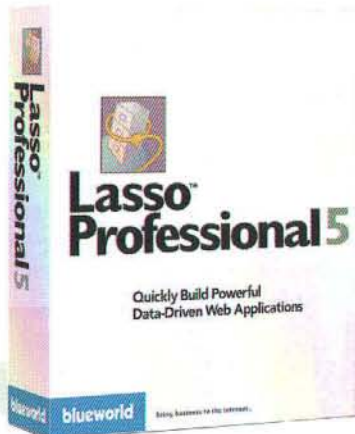
"Awesome!"

"The power, quality and feature set of Lasso Professional 5 is very impressive. The new documentation is definitely among the best in the business."

Johan Sölve; Halmstad, Sweden

"Blue World has managed to add an immense amount of functionality and scalability without sacrificing any of the ease-of-use of previous versions. New features like LassoApps, custom tags and LassoScript create a development environment that seems almost limitless."

Tom Wiebe; Vancouver, Canada



"With Lasso, I have been able to single-handedly develop sophisticated solutions for internal and external use in less time than I see teams of people take with other middleware languages."

Greg Willits; Santa Ana, California

"This new release, I have got to admit, is truly amazing. The rich array of new features, the brand new documentation and the amazing new administration interface make Lasso Professional 5 definitely worthy of a purchase."

Brian Olsen; Brooklyn, New York

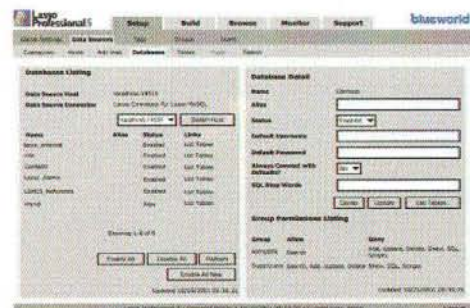
Upgrade today to the most powerful Web application server for Macintosh and beyond and you'll discover why so many Web developers claim Lasso Professional 5 is the must-have tool for quickly building and serving powerful data-driven Web sites.

Lasso Professional 5 introduces a next-generation, object-oriented Web programming language, advanced Web application server administration, an embedded Lasso MySQL™ high-performance database server, a new distributed architecture, new platform and data source support, unprecedented extensibility and customization, 1800 pages of rewritten documentation and over 200 new features and enhancements. Lasso Professional 5 provides vast new power and features while maintaining the legendary ease-of-use, performance, and reliability that make Lasso the preferred tool for tens of thousands of Web developers, ISPs, and IT/IS professionals.

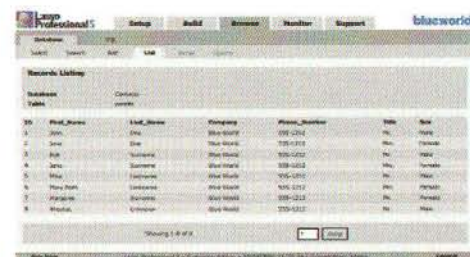
If you're serious about building and serving data-driven Web sites, but don't want to spend a serious amount of time getting it all to work, there's no better choice than Lasso Professional 5.

Visit the Blue World Web site today and see how Lasso products can help you quickly build and serve powerful data-driven Web sites.

Lasso - The Leading Web Tools for Macintosh and Beyond



Lasso Administration controls your entire setup via an attractive and intuitive Web browser interface.



Lasso Database Browser provides instant access to all your databases, without writing a line of code.

blueworld

www.blueworld.com

Another component involved with transfer management is the USB Driver, which is involved with bus access mediation, primarily during device attachment and normal transfers. During attachment, the USB Driver is involved with making sure that a desired configuration will actually be able to function on a given bus. If the configuration can be used, then the USB Driver creates the appropriate pipes for the different transfer types, and deals with things like bandwidth allocation, etc. USB Driver characteristics are usually heavily OS - dependent.

The HCD ensures that all IRPs are tracked correctly, and that bus bandwidth and (micro)frame time maximums are never exceeded. The HCD maintains the transaction list for the various IRPs on a given pipe, and handles client notification of completed IRPs. The transaction list is simply a list of all outstanding transactions for a USB. Each entry in the list contains the transaction description items, such as data sizes, device address, endpoint numbers, etc.

The host controller uses the HCD's transaction list to create the appropriate transactions on the USB. The host controller also provides the reporting mechanism for transaction status tracking. The host controller ensures that various bus access rules are obeyed, such as timeouts, inter-packet timings, etc, and works with the HCD to determine if a new pipe can access the USB.

Another bus management issue is proper handling of Isochronous transfers. Remember that these transfers are time dependent, and have no error correction, so ensuring reliability is important. One of the key components in this is the device itself. It has to provide the device descriptors so that the isochronous transfers it will need can be set up correctly. Because there is no error retry in an isochronous transfer, both ends of the transfer must be both time and data - synchronized so that robust delivery of the data is achieved. Normally, this is done via a well - defined clock system.

Because USB is implemented on such a wide array of hardware and software, a single clock specification wouldn't work. Instead, USB defines a clock model that various systems can adhere to in the best way for that system. There are multiple clocks provided by the USB to help with this. The first is the Sample Clock, which determines the natural data rate of sample moving between the host's client software, and various functions. The Bus Clock is either a 1KHz clock on full - speed segments, or 8KHz clocks on high - speed segments. In a USB, the bus clock is usually a lower frequency than the sample clock. The third clock used is the Service Clock, and this is determined by the rate the client software runs at to service accumulated IRPs.

To ensure that a broad range of isochronous transfers can be handled, there are often multiple samples used so that the transfers aren't interrupted for every sample. The multiple samples are negotiated ahead of time, so that the host controller can set up the most appropriate one for use. Although the bus clock is intended to act as a master clock, real-time data needs make locking to this clock impossible. However, since the data on the bus is packetized, it is easy to just move one unit of data once per (micro)frame, as the (micro)frame clock is reasonably precise.

The USB also provides a frame work for isochronous devices that defines synchronization types, (such as asynchronous, with data rate feedback, Synchronous, locked to the USB SOF rate, and Adaptive, using feedback, or data rate information), methods for endpoints to provide data rate feedback, and connectivity requirements. Endpoints are classified here by their capability to synchronize their data rate to the data rate of the endpoint at the other end of the connection. The endpoints provide feedback by indicating, relative to the SOF frequency, what their data rates are.

If a device is asynchronous, it cannot be locked to either the SOF rate, or any of the USB clocks. Therefore, they clock at either a fixed data rate, one of a limited set of data rates, or a single, continuously programmable data rate. Asynchronous devices must report their programmatic capabilities during endpoint initialization. Their data rate is then locked to either an external clock, or a free-running internal clock outside of the usual three USB clocks. These devices still have to deal with data rate matching, but it is done elsewhere in the USB environment. The asynchronous devices carry their data rate as an implicit part of their sample rate, and must provide explicit feedback information to adaptive drivers.

Synchronous devices can have their clocks controlled via SOF synchronization. These endpoints slave their sample clocks to the 1ms SOF clock tick via a programmable phase - lock - loop, (PLL). Like asynchronous devices, synchronized endpoints can run at a single fixed data rate, one of a limited number of data rates, or a continuously programmable data rate.

Adaptive devices are the most flexible isochronous endpoints. They can send data at any rate in their operating range, and synchronize via constant communications on the bus. The data rate is embedded in the data stream, and can be adjusted on the fly during the transmission.

CONCLUSION

Well, there is a lot of information about USB here, and I haven't even come close to dealing with all of it. To get all of the information, including the source documents I used for this article, please go to <http://www.usb.org/> There is a wealth of information there that is of use to anyone wanting a greater understanding of USB 2.0 While USB 2.0 is not nearly what the people hyping it say it is, (and what ever meets its own hype?), it is a very capable upgrade to a solid standard, and should be a very capable partner along with FireWire in making connecting devices to our Macs faster and easier.

BIBLIOGRAPHY AND REFERENCES

As already mentioned, you may wish to include a bibliography at the end of your article, especially if you wish to cite work in the body of the article. Here is some typical bibliography format, to serve as a model:

Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Phillips. "Universal Serial Bus Specification Revision 2.0". April 27th, 2000.

By Bill Cheeseman, Quechee, VT

AppleScript Studio: Implementing an Application Preferences System

Saving and Retrieving Application Preferences in AppleScript Studio

We provided an overview of AppleScript Studio in the first article in this series, *AppleScript Studio: An Introduction*. In the second and third articles, we will work with Sir Arthur Conan Doyle to solve the mystery of how to create, open and save documents in AppleScript Studio. The existing documentation does not cover this, but an important clue suggesting that it can be done is already in the evidence room: one of the three standard AppleScript Studio templates is named "AppleScript Document-based Application." Although the Watson example that forms the basis of the AppleScript Studio tutorial uses this template, neither it nor any of the other examples actually addresses the mystery. In the course of our detective work, we will therefore help Sir Arthur to write an example application of his own. We will name it "Doyle" in his honor, carrying the current fad of naming applications after Sherlock Holmes characters to its logical conclusion.

The Doyle application will need a preferences file. The first solution an AppleScripter thinks of when the subject of preferences comes up is to use a property. However, the *Release Notes* for the first version of AppleScript Studio mention that AppleScript properties do not retain changes to their values between relaunches of the application. This is contrary to the normal behavior of AppleScript properties (and, for that matter, of AppleScript global variables, which also retain their values across repeated launches of a script or script application). A future release of AppleScript Studio will presumably restore the persistence of AppleScript properties. In the meantime, the AppleScript Studio *Release Notes* recommend that "if you want persistent storage of values, write them to a preferences file." We will therefore include in Doyle the ability to create, open and save a preferences file, which provides a simple introduction to the use of documents in AppleScript Studio. The exercise will

prove valuable even if properties eventually resume their accustomed behavior, since preference files can be useful in any application of even modest complexity.

We will tackle the mystery of documents in two installments. In the first installment, we will devise an AppleScript-based preferences system in response to the injunction in the *Release Notes*. In the second, we will expand the example application to handle more complex documents.

Unlike Sir Arthur's novels, the current mystery does not have only one solution. There are many ways to save and retrieve information in AppleScript Studio, as in Cocoa applications generally. This article offers one solution among many.

This article and the next will include complete listings of the AppleScript statements in the Doyle application. In order to keep the scripts as simple as possible, we will make no attempt to deal with internationalization and localization, and we will do no error trapping.

THE APPLESCRIPT DOCUMENT-BASED APPLICATION TEMPLATE

The first step in writing any AppleScript Studio application is to create a new project in Project Builder, using its **File -> New Project** menu item to select one of three standard templates. Do this now, and choose the AppleScript Document-based Application template. Save our new project under the name "Doyle" wherever you keep your scripting projects; your **Documents** folder will do.

Before deciding what we need to add to the template, it will be useful to build and run the template as is, without any changes, to see what already works. Do this now by clicking the **Build and Run** button in Project Builder. An empty window named "Untitled" appears. This is in accord with the *Aqua Human Interface Guidelines*, which specify that launching an application or bringing a running application to the front (for example, by clicking its icon in the Dock) should open a new, untitled window if a window belonging to the application is not already open, and it should deminiaturize a window if the only

Bill Cheeseman is a retired lawyer now making his living as a Macintosh developer and loving every minute of it. He is uniquely qualified to write about AppleScript Studio, having served as webmaster of The AppleScript Sourcebook (www.AppleScriptSourcebook.com) for many years and having also written a Cocoa tutorial, *Vermont Recipes – A Cocoa Cookbook* (www.stepwise.com/Articles/VermontRecipes/).

THE MISSING LINK!

Don't throw out your old machine or peripheral,
just find the right adapter!



K3 Smart Card Reader for USB

(with optional developer kit –
let your App read Smart Cards!)

\$184⁹⁵

2nd Wave FireWire to SCSI Converter



Connect SCSI Devices to
your FireWire Mac!

\$78⁹⁵

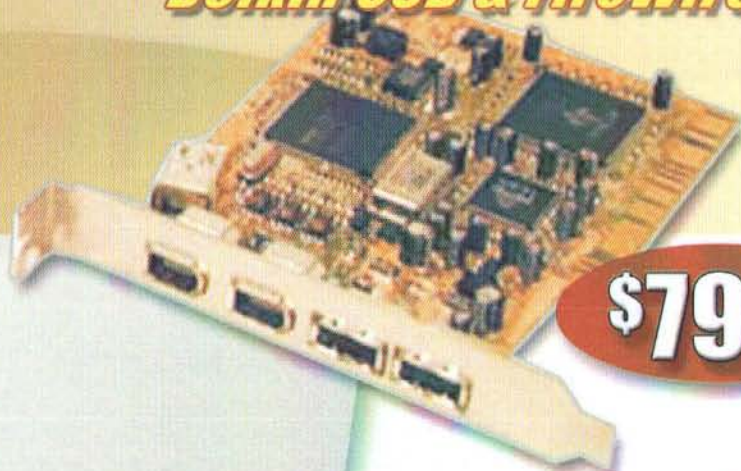
DevDepot your source for connectors and adapters of all kinds!

DEV DEPOTSM

www.devdepot.com/connectors

PO Box 5200 • Westlake Village, CA • 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)

Belkin USB & FireWire Combo PCI Card



Get two for one, add the power of FireWire and the flexibility of USB to your PCI Mac with one card!

\$79⁹⁵

DVIator

Connect an Apple flat panel display to your DVI Mac!



\$147⁹⁵

KEYSPAN Twin Serial

Turn one USB port into two serial ports!



\$69⁹⁵

Griffin iMate

Connect ADB devices to your USB Mac!



\$34⁹⁵

Griffin iMic

Add audio in and out to your USB Mac!



\$32⁹⁵

open window is currently miniaturized in the Dock.

The new window contains no controls, but our application does have a menu bar, so let's try out some of its menu items. **About Application**, under the **Application** menu, brings up an About dialog. Like several of the menu items, the About dialog contains dummy information. We will shortly customize the menu items and the About dialog using Interface Builder, but we won't have to write any scripts to make them work. **Help -> Application Help** brings up a dialog stating that the application doesn't have any help. We'll leave that alone in these two articles.

The menu that interests us in these articles is the **File** menu, with its built-in **New**, **Open**, **Close**, **Save**, **Save As**, and **Revert** menu items. Try them out. Choose **File -> New**; a new empty window appears, named "Untitled 2." Choose **File -> Close**; the new window closes, leaving the original "Untitled" window in place. Choose **File -> Save** or **Save As**; a standard file saving sheet opens with all the usual features, allowing us to give the document a name and select a location where it should be saved. This is getting worrisome, for it is beginning to appear that there may be no mystery for us to solve!

Continuing with the investigation, name the untitled window "Test" and select your **Documents** folder as the place to save it. Another sheet promptly appears, saying "Couldn't Save Document." "Aha!" says Sir Arthur, "Here's a mystery. Holmes will have to write a script or two to save this document."

LAYING THE GROUNDWORK

Before figuring out how to save the document, we will have to help Sir Arthur decide why Holmes would want to save it in the first place, by specifying a task that requires saving and retrieving information. Let's have Doyle (the application) save a log documenting the course of the investigation, recording the date and time of every occasion when a user launches or quits Doyle. To make it more useful, we'll also allow the user to open the log and edit it by adding comments to each entry. In case some detectives prefer to let the evidence speak for itself, we'll also give Doyle a Preferences window in which the ability to edit the log can be turned on and off. It is this last feature, the Preferences window, that we will tackle in this article, leaving the Doyle Usage Log to the next installment.

The basics of building an AppleScript Studio application are covered well in the documentation and need not be repeated here except in brief outline.

First, customize the menu bar. With our new Doyle project open in Project Builder, mouse over to the Groups & Files pane on the left side of the project window, selecting the Files tab if necessary to open it. Expand the Resources group, and double-click **MainMenu.nib**, one of the files that comes with the template. Interface Builder launches and opens the new application's main menu. Following the steps outlined in the AppleScript Studio documentation, change several of the menu items so they refer to "Doyle" instead of "Application." Don't change the name of the **Application** menu itself, however; it will automatically take on

the name of our application when we run it. Don't forget to save frequently and, when you're done, close the **MainMenu.nib** file by clicking the **MainMenu.nib** window's close box.

Next, customize the About box. Click **Credits.rtf** in the Resources group of the Groups & Files pane in the Project Builder window, and type any appropriate information identifying the team that built Doyle.

Finally, click **InfoPlist.strings** in the Resources Group and provide new values for the four variables you find there, following the model of the dummy strings in the template. Basically, just change "Application" to "Doyle" wherever it appears, and change the name of the author in the copyright notices so that some fake company doesn't get credit for our work.

DESIGNING THE WINDOWS

The Doyle Usage Log Window

Although the details of implementing the Doyle Usage Log will be left to the next installment, we will design the main document window now to give us a sense of our application's look and feel. Double-click **Document.nib** in the Project Builder window's Resources group to open it in Interface Builder. If you don't see the empty document window at first, double-click the Window icon in the Instances pane of the **Document.nib** window to bring it to the front. Also, for convenience you can open the Info panel now by choosing **Tools -> Show Info**. It is very helpful to have this panel open all the time; for this reason, an Interface Builder preference exists to **Show Info Window at Startup**, but it is turned off by default.

With the document window open and selected, use the Attributes pane of the NSWindow Info panel to set the window's title to "Doyle Usage Log." Also, select buttons and checkboxes in the Info panel as needed to set the window's Backing to **Buffered**, its Controls to enable **Miniaturize**, **Close** and **Resize**, and its Options to enable **Visible at launch time**, **Deferred**, and **One shot**. The meaning of most of these settings is obvious from their names. You will rarely, if ever, need to change the default settings of those whose meaning is not obvious.

From the Cocoa-Data palette, drag a table view into the document window and place it near the top left corner, using the Aqua guides to position it precisely in accordance with the *Aqua Human Interface Guidelines*. Drag the table view's bottom right resize handle to make the table fill the window to the right Aqua guide and far enough from the bottom to leave room for a row of buttons. There is no Aqua guide to leave room in advance for the buttons; you will have to readjust the table view in a moment, after adding a button. Create a third column in the table view and name the columns, from left to right, "Date," "Action," and "Comments," using techniques described in the AppleScript Studio documentation. Also, size the left column to hold a date and the center column to hold the word "Run" or "Quit." Finally, select checkboxes in the Info panel to set the table view's Selection to **Allows Empty Selection**, **Allows Multiple Selection**, and **Allows Column Selection**, its Scrollbars to **Vertical** and **Horizontal**,

and its Options to Allows Resizing, Display Column Headers, and Autoresizes Columns to fit. Other items should be deselected, including Allows Reordering because we want the log to remain in chronological order.

Drag a standard push button from the Cocoa-Views palette to a position near the bottom right corner of the document window, lining it up with the aid of the Aqua guides. Name the button "Delete." The button may resize itself to accommodate the length of its new title, so you might have to readjust its position afterwards. You can also now drag the bottom edge of the table view to position it the proper distance above the button, using the Aqua guides that have now become available.

The Preferences Window

Now we are ready to design the Preferences window. Close Document.nib to get it out of the way for now by clicking its close box, saving it if asked. Still in Interface Builder, choose File -> New; the Starting Point dialog opens, if it wasn't already open. Select Empty under the Cocoa topic and click the New button. A new, untitled Interface Builder nib window opens, with the Instances pane selected and containing File's Owner and First Responder icons. For convenience, save it now, before making any changes, by choosing File -> Save As. Give it the name "Preferences.nib" and navigate to save it in the English.lproj subfolder of the Doyle project folder. Leave the checkbox unselected so that the file extension will not be hidden. When a sheet appears, click the Add button to add the new nib file to the Doyle project and target. If this sheet does not appear, you saved the nib file in the wrong place; this can easily be cured by using Project Builder's Project -> Add Files command, preferably first dragging the new nib file into the correct location using the Finder in order to keep all the project files in one place. After it appears in the Groups & Files pane of the Project Builder project window, it is convenient to drag it into the Resources group if it isn't already there. The small checkbox to its left should be selected, indicating that the file will be included in the target when the application is built.

Select the File's Owner icon in the Instances tab of the Preferences.nib window and, in the Attributes pane of the File's Owner Info panel, select NSApplication to make the application object the owner of this nib file. If a dialog appears warning you that something evil will happen, you haven't succeeded in adding the Preferences.nib file to the project. You can proceed with the design of the window anyway, but be sure you remember to add the file to the project later using Project Builder's Project -> Add Files command.

Drag a window icon from the Cocoa-Views palette into the Instances pane of the Preferences.nib window. A Window icon appears in the Instances pane and an empty window appears on the screen. Using the Attributes pane of the NSWindow Info panel, give the window the title "Preferences" and configure its settings as you did with the document window, with these two exceptions: the Resize checkbox in the Controls area should be deselected, as should the Visible at launch time checkbox.

Finally, drag a checkbox (or "switch," as it is sometimes known in Cocoa) from the Cocoa-Views palette to the top left corner of the Preferences window, and rename it "Show Comments in Usage Log." Then, resize the window to make it smaller, until the right Aqua guide indicates that you have the correct margin on the right and until the bottom edge is as close to the top as you are allowed to drag it. With the checkbox still selected, use the NSButton Info panel to select the Selected checkbox in the Options area. We will set the Show Comments in Usage Log preference to true by default in one of Doyle's scripts in just a moment.

Save the Preferences.nib file but leave it open. We are now ready to start scripting.

SCRIPTING THE APPLICATION

We will start by scripting the Preferences window, because it reflects a single value that must be read by the main document window before the latter is opened. We will set up the Preferences window and scripts for setting and getting its one preference value first. In subsequent sections, we will take care of coordinating this preference value and the user interface, and we will arrange to save this value to disk in a preferences document and retrieve it when the application is launched.

Managing the Preference Value

We now have a Preferences window that will allow the user, by checking or unchecking a checkbox, to specify

DO YOU NEED TO "MODERNIZE" YOUR SEARCH TECHNOLOGY?



POWER TOOLS FOR YOUR WEBSITE:

Onix - Search and Retrieval Engine
RouteX - Document Filtering and Routing Engine
Brevity - Document Summarization Engine

<http://www.lextek.com>

whether comments are to be shown in the Doyle Usage Log. Next, we need to provide a means to hold this preference value—true or false—within the application, so that it can be looked up every time the user opens the Log. An easy way to do this would be to rely on the state of the checkbox itself as a record of the state of the preference item, but we won't do it that way. For many reasons, data values should almost always be stored separately from the user interface. Here, we will save the value of the application's single preference item in a disk-based preferences file.

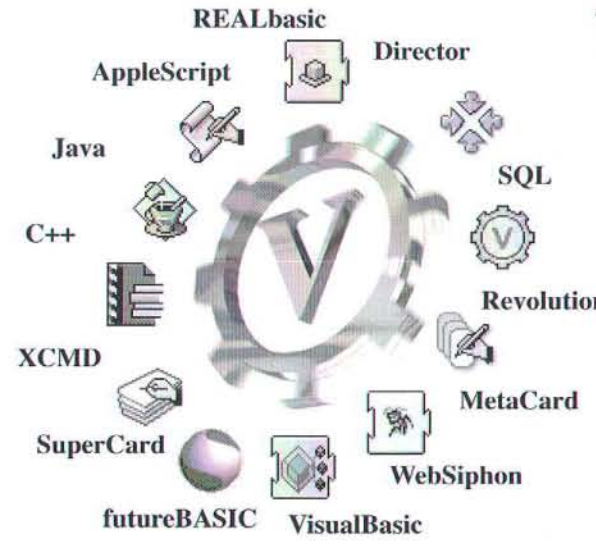
We will not use Cocoa's built-in application preferences system, based on the `NSUserDefaults` class, but will instead store Doyle's preferences as an AppleScript record using the Read/Write commands implemented in the Standard Additions scripting addition. In the first release of AppleScript Studio, the Cocoa application preferences system is not available to AppleScript. The AppleScript Studio engineering team has informally made available a temporary workaround for using it, in the expectation that AppleScript access to Cocoa's `NSUserDefaults` class will become a built-in feature in a later release. Even then, however, there may be reasons to prefer a pure AppleScript solution, so we will develop one here.

As the AppleScript Studio documentation very briefly suggests, it is generally a good idea to follow the Model-View-Controller, or MVC, design pattern, whereby the Model (consisting of the application's data and data management algorithms) is kept separate from the View (the application's graphical user interface). The MVC paradigm is particularly important in Cocoa applications, because the Cocoa frameworks are built around the concept and depend upon it in a variety of ways. It may be less important in an AppleScript Studio

application, where the event handlers are already laid out according to the AppleScript object model and the structures of a Cocoa application. We will nevertheless follow the MVC paradigm here by storing and manipulating our single preference value in one script, called `PreferencesModel.applescript`, while coordinating the preference item and the user interface in a second script, `PreferencesController.applescript`.

If, after reading this article, you adopt this system and create an application of your own having many preference items, you will be thankful that you started out this way. It will allow you to keep all of the preference item data and handlers to manipulate it in one script, while isolating handlers to update the user interface and to obtain information from the user about the preference items in a separate script. Although it isn't necessary to create separate scripts in this manner in AppleScript Studio, getting in the habit of breaking scripts into smaller pieces according to some reasonable plan will help you to keep the scripts manageable as your application grows larger. Here, if you later change the user interface, you will at most have to revise the Controller script; the Model script will continue to work without change. Likewise, if you later rearchitect the data storage strategy, you will at most have to revise the Model script but can leave the Controller script and the rest of the application alone.

Our strategy for implementing these two scripts will be dictated by the fact that AppleScript properties and global variables do not retain their values across relaunches of the application, but the strategy will work as desired even if this problem is repaired in a subsequent version of AppleScript Studio. We will save our preference item to a disk file immediately after it is changed by the user and retrieve it



Valentina

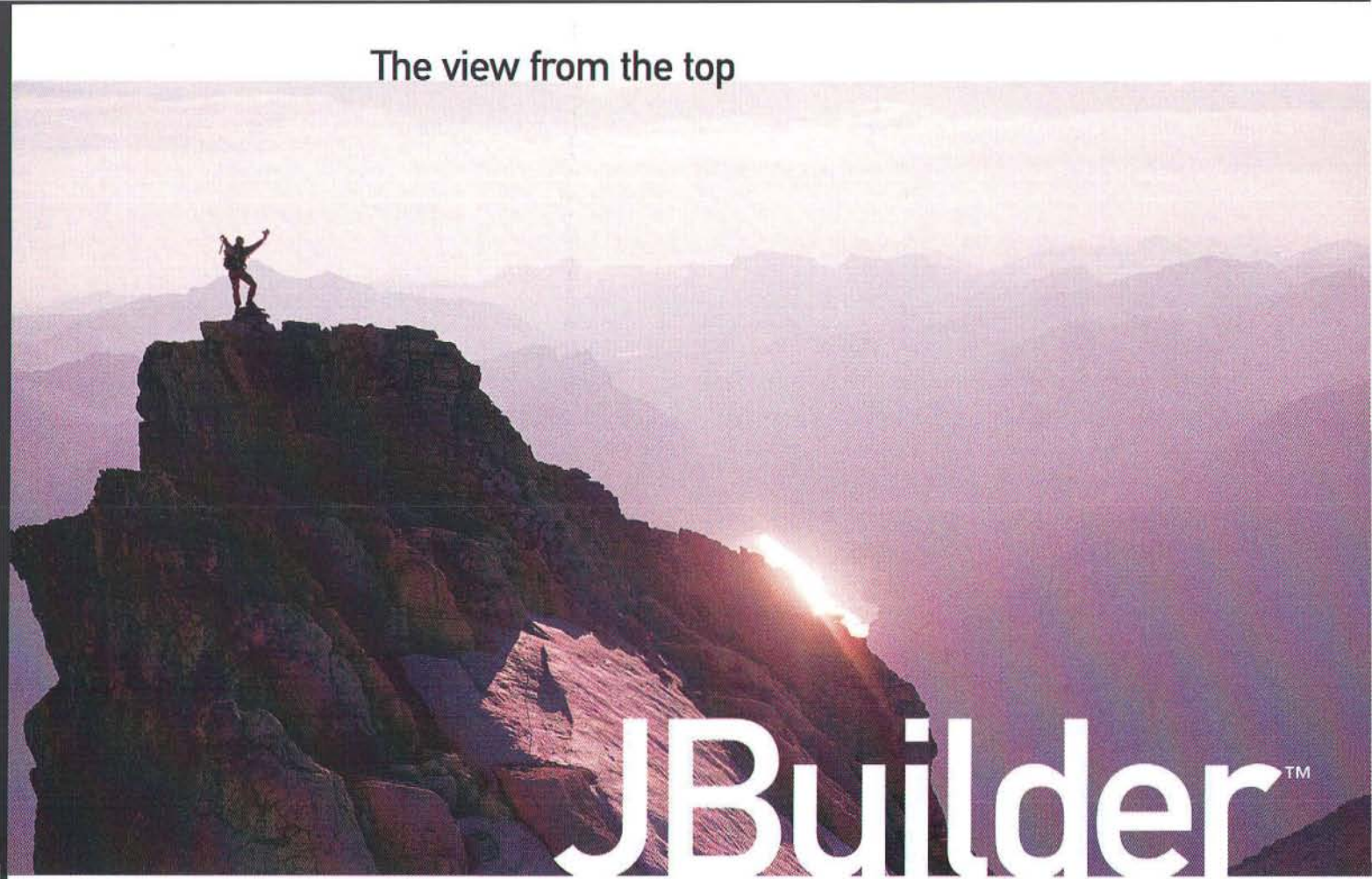
Object-Relational SQL Database

**The fastest database engine
for MacOS/Windows**

**It operates 100's and sometimes
a 1000 times faster than other systems**

www.paradigmasoft.com
 Hosted by macserve.net
 Download full featured evaluation version

The view from the top



JBuilder™

Borland®

The possibilities are limitless with JBuilder, the leading Java™ development solution.

Take EJB development to the next level with an intuitive visual designer. Rapidly develop and deploy J2EE™ e-business applications to multiple application servers including Borland® Enterprise Server, BEA® WebLogic®, IBM® WebSphere®, and iPlanet™ Application Server. Develop and deploy applications on Windows®, Linux®, Solaris,™ and Mac® OS platforms. Efficiently collaborate as a team with support for leading version control systems. Build data-driven Web applications using servlets, JSP,™ and XML.

View code in a dramatic new way with UML code visualization. Experience extreme productivity with refactoring, unit testing, and documentation tools.

See for yourself at <http://www.borland.com/new/jb6/5064.html>



from disk whenever it is needed. Every time the application is launched, it will look for a preferences file in a standard location. If it does not find the file there, it will immediately create a new one, setting its contents to initial default values coded into the application. If it does find a preferences file, it will use the settings found in it, even if the settings it used the last time it was run were different (for example, the user might have substituted a new preferences file for the old file in the meantime).

This strategy will be implemented in a new script, `PreferencesModel.applescript`. The script will include several handlers for opening the preferences file, closing it, setting its initial default values if the file does not already exist, and retrieving all of its values. In addition, it will include, for the one preference item implemented in this article, a “get” and a “set” handler to retrieve or change its value. Additional get and set primitives can be added later, if additional preference items are needed. These are termed “primitive” handlers, because they get down and dirty with the underlying details of our preferences system. Later, we will call these primitive handlers from higher level handlers implemented in the `PreferencesController.applescript`, to interact with the user.

We will set the initial, or default, value of our one preference item to true in a permanent property, `preferencesDefaultRec`, to match the selected state of the checkbox that we created earlier in the Preferences window. The property holds an AppleScript record, and additional labeled fields can be added to it if additional preference items are needed. This default value will be used whenever the application can't find a preferences file. Strictly speaking, it wasn't necessary to select the checkbox in Interface Builder, as we did above, because the application will get the default setting from this property in any event and update the checkbox to match. The property that holds the default preference item value will never be changed by our code, so it will be initialized to the original default value every time the application is launched, and the lack of persistent property changes in this release of AppleScript Studio won't matter. If we change the default, we will recompile `PreferencesModel.applescript`, and its new value will thereafter remain as is until we again recompile.

As described above, we will provide primitive handlers in `PreferencesModel.applescript` to initialize and to get and set the values of this property on disk. These handlers are intended to be called from another script in the application, `PreferencesController.applescript`. However, handlers in another script cannot be called directly in AppleScript Studio. Instead, we have to resort to a simple strategy of indirection. We place each of them inside an explicit script object defined in `PreferencesModel.applescript` using AppleScript's `Script` keyword. In fact, the entire body of `PreferencesModel.applescript` is enclosed in a script object, named `preferencesModelLib`. Any other script in our application that needs to call one of these handlers will use the `Load Script` command from the Standard Additions scripting addition to load a copy of

`PreferencesModel.applescript` and store its script object, `preferencesModelLib`, into a property, `prefsModelLib`, declared in the calling script. Then the handlers of the loaded script object can be called by telling the property in the calling script to execute any of them.

Declaring a script object within a script, as we have done here, is sometimes required in AppleScript, for technical reasons having to do with the context within which the script's handlers are to be compiled and executed. For a detailed explanation, read the Script Objects chapter of the *AppleScript Language Guide* and a short article I wrote for The AppleScript Sourcebook, *Subroutine Handlers in Script Libraries and Script Servers*, at www.applescriptsourcebook.com/tips/scriptserver.html.

To create the new script, select the Preferences window in Interface Builder and go to the AppleScript pane of the NSWindow Info panel. Click the New Script button at the bottom and save it as `PreferencesModel.applescript` in the Doyle project. New AppleScript scripts can be created in this manner in Interface Builder or, as we will see later, in Project Builder.

Now switch to Project Builder, where you will see the new `PreferencesModel.applescript` file in the Scripts group. Select it, and, in the editing pane of the project window, type the script object and handlers shown in Listing 1. In AppleScript jargon, this file is called a “script library.” The use of script libraries is a common, if somewhat advanced, technique in AppleScript generally. A script library functions much like an “include” file, allowing properties, handlers and other AppleScript constructs declared in one file to be used in another file through AppleScript's `Load Script` mechanism.

Listing 1: PreferencesModel.applescript

(* PreferencesModel.applescript *)

(* This is an MVC Model script that manages the values of preference settings. It provides an initialization handler for all of them and get and set accessors for each, as well as some utility handlers. *)

(* Initial default values are set in the `initPreferences()` handler, which is executed only if no preferences file is found at launch. Otherwise, the values of all preferences are obtained from the preferences file. A preferences file is saved in the current user's Preferences folder at `~/Library/Preferences`. The file is always closed immediately after reading or writing in order to avoid corruption in the event of a crash. Preferences are saved and managed as a single AppleScript record. Error checking is omitted. *)

(* Properties *)

— Name of preferences file
property preferencesFileName : “Doyle Preferences”

— Default preference record; this property may not be changed by a user, so it serves
— as a permanent record of default values for initialization of new preferences files
property preferencesDefaultRec : {showCommentsPref:true}

— Insert other fields in the `preferencesDefaultRec` record with comma delimiters
— if additional preferences are added, and recompile.

(* Script Objects *)

script preferencesModelLib

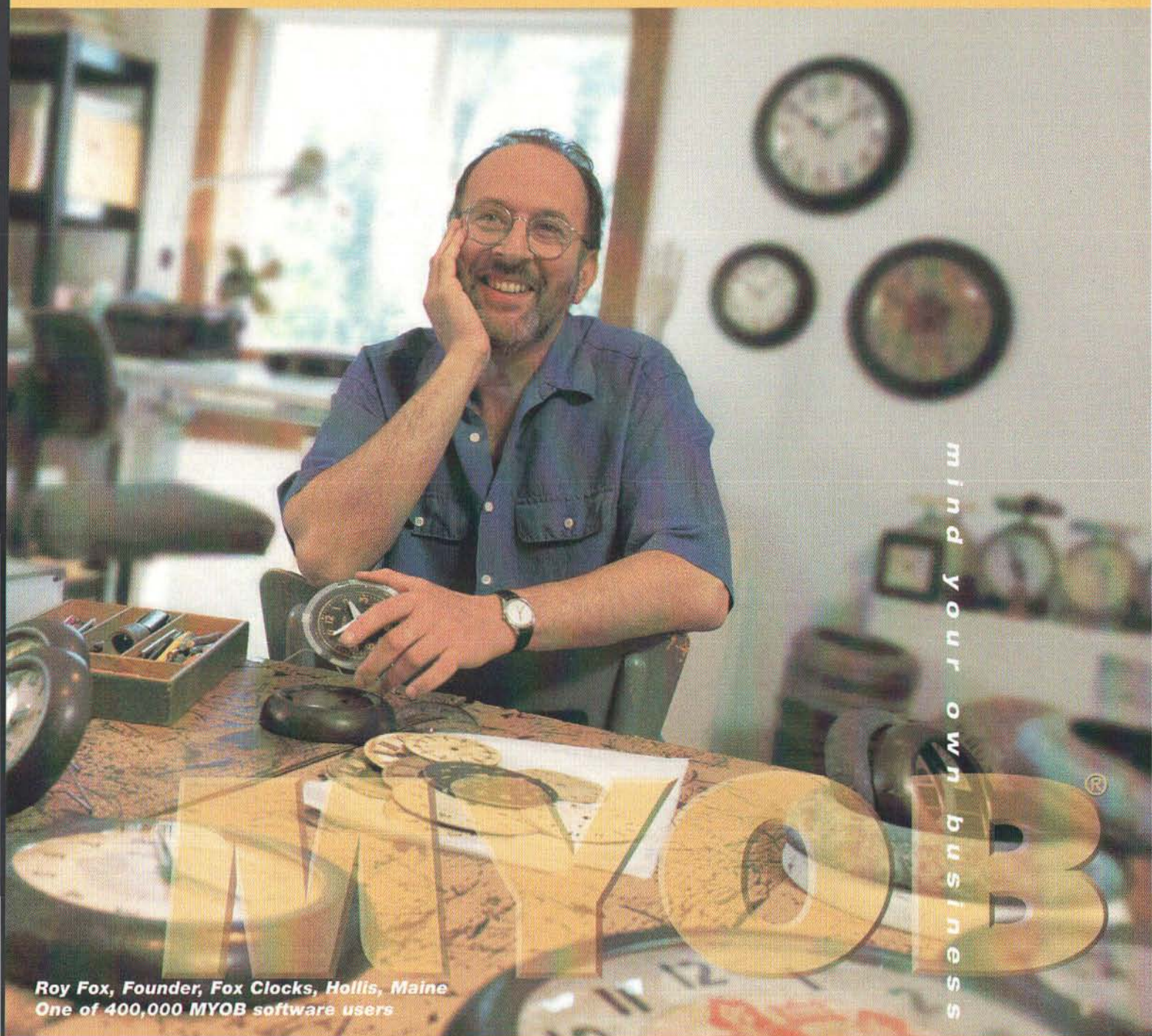
(* Handlers *)

— These handlers provide access to preferences at the most primitive level.
— Load this script object into another script and call these primitive handlers

DESIGNED FOR THE WAY YOU WORK

I'd rather create clocks than invoices.

If I wanted to keep books all day, I'd have been an accountant.



mind your own business

MYOB

Roy Fox, Founder, Fox Clocks, Hollis, Maine
One of 400,000 MYOB software users

MYOB

**Small Business.
Smart Solutions.SM**

AccountEdge on the Mac.
Software includes QuickBooks[®] conversion
utility. MYOB also offers conversion service.

800.322.MYOB
myob.com/us

[®]QuickBooks is a trademark of Intuit, Inc.

MYOB software is the simplest, most powerful, most complete
solution for managing my company on the Mac, from the day to day
to the bottom line.

Antique frames. Quartz movements. That's my business.

MYOB software works for me.

- wherever access to preferences is required. If the preferences infrastructure
- is later changed, only this script will require revision, so long as the
- names and return values of these handlers remain unchanged.

```

on openPreferences()
    — Opens current user's preferences file, creating an empty file if none exists.
    — Returns path to current user's preferences file for use in other handlers.
    set preferencesFilePath to ~
        (path to preferences from user domain as string) ~
        & preferencesFileName
    open for access file preferencesFilePath ~
        with write permission
    return preferencesFilePath
end openPreferences

on closePreferences()
    — Closes current user's preferences file, saving changes.
    set preferencesFilePath to ~
        (path to preferences from user domain as string) ~
        & preferencesFileName
    close access file preferencesFilePath
end closePreferences

on initPreferences()
    — Sets preferences to default values if no preferences file exists.
    set preferencesFilePath to openPreferences()
    try
        read file preferencesFilePath
        — discard if successful
    on error number -39
        — end of file indicates file is empty (just created)
        write preferencesDefaultRec to ~
            file preferencesFilePath as record
    end try
    closePreferences()
end initPreferences

on getPreferencesRecord()
    — Returns the entire current preferences record.
    set preferencesFilePath to openPreferences()
    set thePreferencesRec to ~
        read file preferencesFilePath as record
    closePreferences()
    return thePreferencesRec
end getPreferencesRecord

on getShowCommentsPreference()
    — Returns current value of show comments preference item.
    set thePreferencesRec to getPreferencesRecord()
    return showCommentsPref of thePreferencesRec
end getShowCommentsPreference

on setShowCommentsPreference(setting)
    — Sets new value of show comments preference item.
    set preferencesFilePath to openPreferences()
    set thePreferencesRec to ~
        read file preferencesFilePath as record
    set showCommentsPref of thePreferencesRec to setting
    set eof file preferencesFilePath to 0
    — discard old contents of file
    write thePreferencesRec to ~
        file preferencesFilePath as record
    closePreferences()
end setShowCommentsPreference

— Insert additional accessors on pattern of getShowCommentsPreference() and
— setShowCommentsPreference(setting) here if additional preferences are added.

end script

```

The odd character at the ends of some lines in Listing 1 is AppleScript's line continuation character, allowing us to split long lines into shorter segments so they don't run off the page to the right. In most script editors, it is possible to type the line continuation character and start a new line in one stroke by pressing Option-Return. In this release of AppleScript Studio, however, you must type Option-L followed by Return.

After you have typed the body of `PreferencesModel.applescript`, examine each of its handlers. These are garden-variety AppleScript statements, using the Read/Write commands in the Standard Additions scripting addition. If you aren't familiar with them, you can read about them in Apple's *Scripting Additions Guide* or any of the several good books about AppleScript currently in print.

Updating the User Interface

Next, we need to write the `PreferencesController.applescript` script that will call the primitive handlers in `PreferencesModel.applescript` to get and set the preference item's value, display it in the Preferences window, and accept changes made by the user.

To carry out the Model-View-Controller (MVC) paradigm consistently, we will use this new script to hold the handlers that coordinate the user interface of the Preferences window with the value of the preference item. As its name indicates, it is a Controller, the third component of the MVC trinity. The second component, the View, is essentially already written, in that AppleScript Studio comes with many predefined view classes, such as windows and user controls, with built-in event handlers to take care of their basic functionality. By putting handlers that control the state of the various View objects into a separate Controller script, we effectively isolate the View from the Data. The Controller serves as an intermediary between the Model and the View and therefore must know about both of them. `PreferencesModel.applescript`, which is the Model holding the preference system's handlers for accessing the data, knows nothing about the nature or features of the user interface (the View), and the built-in Cocoa View objects know nothing about the structure and implementation of the data they represent (the Model). For that reason, `PreferencesModel.applescript` need not be changed in any way even if we completely revise the user interface. Only the Controller script will need to be customized if the user interface is changed.

Go into Project Builder now and create the new script. Choose File -> New File, then in the New File assistant select AppleScript File under the AppleScript heading and click Next. In the New AppleScript File assistant, name it `PreferencesController.applescript` and click the Finish button, first making sure that the location is set to the Doyle project folder. If the new file appears elsewhere in the Groups & Files pane of the project window, drag it into the Scripts group. Type the property and script object shown in Listing 2 into `PreferencesController.applescript`.

Listing 2: PreferencesController.applescript

(* PreferencesController.applescript *)

(* This is an MVC Controller script that contains handlers to coordinate data values in PreferencesModel.applescript with the user interface in the Preferences window. *)

(* Properties *)

property prefsModelLib : null

OPENBASE SQL 7.0

(* Script Objects *)

script preferencesControllerLib

(* Handlers *)

- These handlers get and set preferences using plain English terminology, by loading and calling the primitive handlers in PreferencesModel.applescript.
- If the preferences infrastructure in PreferencesModel.applescript is later changed, this script will not require revision, so long as the names and return values of the primitive handlers remain unchanged.

```
on initPrefs()
    loadPreferencesModelLib()
    tell prefsModelLib to initPreferences()
end initPrefs

on commentsAreShown()
    loadPreferencesModelLib()
    tell prefsModelLib to ^
        return getShowCommentsPreference()
    end commentsAreShown

on showComments()
    loadPreferencesModelLib()
    tell prefsModelLib to setShowCommentsPreference(true)
end showComments

on hideComments()
    loadPreferencesModelLib()
    tell prefsModelLib to setShowCommentsPreference(false)
end hideComments
```

end script

(* Handlers *)

```
on loadPreferencesModelLib()
    — Loads PreferencesModelLib from PreferencesModel.applescript, if not loaded.
    if class of prefsModelLib is not script then
        load script POSIX file ^
            ((path for script "PreferencesModel" ^
                extension "spt") of main bundle)
        set prefsModelLib to preferencesModelLib of result
    end if
end loadPreferencesModelLib
```

(* Event Handlers *)

— See Listings 5, 6, and 7 for the event handlers in this script.

The AppleScript handlers that are enclosed in an explicit script object named `preferencesControllerLib` in `PreferencesController.applescript` are the high-level handlers we referred to above. Their names implement English-like grammar intended to facilitate their use in simple statements that sound very like normal English sentences. Each of them first loads the script object in `PreferencesModel.applescript`, if it is not already loaded, and then calls the primitive methods implemented there. The high-level handlers are placed in a script object in `PreferencesController.applescript` so that they, in turn, can be loaded and called by other scripts in the application, as we will see later. While this double layering of handlers is not necessary in an AppleScript Studio application, it is used here as part of our effort to separate the Model object from the View in accordance with the MVC paradigm. It echoes a technique commonly used in Cocoa applications.

Outside of the `PreferencesControllerLib` script object, `PreferencesController.applescript` implements one property and

How IMPORTANT IS Your DATA?

With its redundant, multi-file design,

OpenBase SQL

ensures data integrity and availability,

while enhancing database performance.



OPENBASE
INTERNATIONAL

For more info, visit:

www.openbase.com

one handler. The property, `prefsModelLib`, will hold the `preferencesModelLib` script object from `PreferencesModel.applescript`. It is assigned that value by the handler, `loadPreferencesModelLib()`, using the Load Script command from the Standard Additions scripting addition.

The `loadPreferencesModelLib()` handler uses a common AppleScript technique to load the script library only when loading is needed. It first tests the `prefsModelLib` property to see whether it is of the built-in AppleScript class, `Script`. If it is, we know that the script library has already been loaded, because the `prefsModelLib` property was initially set to null in its declaration. This technique takes advantage of the fact that AppleScript properties and variables are not strongly typed; their type is the type of whatever value they currently hold. Therefore, we can call the `loadPreferencesModelLib()` handler in every handler that gets or sets a preferences value, knowing that the call will do nothing if the script is already loaded, in order to ensure that the script will always be loaded when necessary. As a bonus, if properties retain their values across relaunches of an application in a future version of AppleScript Studio, the script library will only have to be loaded the first time the application is run after it has been compiled or recompiled. (The modest execution efficiency this gives us on subsequent launches comes at the expense of some increase in file size. If this is an issue for your application because its preference settings are very large, you can reset the `prefsModelLib` property to null when quitting the application).

The file holding the `PreferencesModel.applescript` script must, of course, be found before it can be loaded in this manner. We know from the AppleScript Studio documentation and from examining built examples that the script files contained in an AppleScript Studio application bundle are located in a `Scripts` subfolder in the `Resources` folder in the application bundle's `Contents` folder. One way to find these scripts, therefore, is to code this path explicitly into a handler. The only other thing we need to know is the path to the running application, which can be obtained from the Standard Additions' `Path To Me` command. Also, the script's file extension will change from `".applescript"` in its text form to `".sct"` in its compiled form. The `pathToPreferences()` handler in Listing 3, using this technique, is cribbed in part from several of the AppleScript Studio examples.

Listing 3

```
on pathToPreferencesModel()
    set appPath to (path to me from user domain) as text
    return (appPath & "Contents:Resources:Scripts:") as text
end pathToPreferencesModel

on loadPreferencesModelLib()
    if class of prefsModelLib is not script then
        set prefsModelLib to load script file ~
            (my pathToPreferencesModel() & ~
              "PreferencesModel.sct")
    end if
end loadPreferencesModelLib
```

There is a more flexible means to accomplish the same end,

however, using special features in AppleScript Studio's dictionary. The Application Suite in `AppleScriptKit.asdictionary` includes a `Path For` event that returns the path to any of several special AppleScript Studio locations, including the location of a script in the application bundle given its name and its compiled file extension. Similar features are provided in the `Bundle` class, whose properties provide the paths to important components of an application bundle. Here, we use `Path For` in conjunction with the application object's `Main Bundle` property, which will hopefully give our application a more robust capability to survive any future changes to the location in which scripts are bundled in some future version of AppleScript Studio.

There is a wrinkle to using the `Path For` event and the similar class properties: they return folder and file paths using the forward slash character as a delimiter, as required by Cocoa, whereas most AppleScript commands, such as the `Load Script` command in Standard Additions, require the colon character as a delimiter. Fortunately, AppleScript 1.8, included with the first release of AppleScript Studio, contains a new `POSIX File` class and a `POSIX Path` property to make it easy to convert file paths between the slash-delimited POSIX form and the colon-delimited AppleScript form.

A step-by-step version of our `loadPreferencesModelLib()` handler using the `POSIX Path` property is set forth in Listing 4. Note that the `Path For Script` command takes a second parameter, `Extension`, which is not documented in the manual or the `AppleScriptKit.asdictionary`. Its usage can be gleaned from one of the examples; using `path for script "Preferences.sct"` will not work. Note also that we do not have to bracket `Main Bundle` in a `Tell Application` block, because this is assumed in AppleScript Studio. (The final version of our `loadPreferencesModelLib()` handler, telescoping the first three lines of the `If` clause in Listing 4 into a single statement, can be seen in Listing 2.)

Listing 4

```
on loadPreferencesModelLib()
    if class of prefsModelLib is not script then
        get (path for script "PreferencesModel" ~
            extension "sct") of main bundle
        get POSIX file result
        load script result
        set prefsModelLib to preferencesModelLib of result
    end if
end loadPreferencesModelLib
```

Connecting Event Handlers

As written to this point, our scripts will do nothing, because none of the handlers we have written has yet been "connected" to an AppleScript Studio event handler. This is a fundamental lesson to be learned in writing AppleScript Studio applications. No AppleScript handlers will be executed unless they are either directly connected to an AppleScript Studio event handler using the AppleScript pane in the Interface Builder Info panel, or unless they are called (directly or indirectly) from a handler that is connected to an event handler in this manner. This is why, for

Special Small Dogs

Here are just a few of the photos our customers have sent us of their special dog friends!



Check out all the other special Small Dog photos and send us one of your own at www.smalldog.com!

Small Dog's Special



PowerBook G4/667 Titanium
\$3199

- 512MB RAM
- 30gb Hard Drive
- Combo Drive
- Airport card and base station



Purchase from an Apple Specialist!

Small Dog Electronics is proud to have earned Apple's prestigious designation of Apple Specialist. Small Dog Electronics has exceeded all of Apple's stringent requirements for its Specialists and has taken it one step further.

Small Dog sells only the most powerful personal computers in the world—every single one of them an Apple Macintosh! In addition to the rigorous Apple training program, each Small Dog employee has chosen a specific area of expertise to concentrate upon. You can be assured that whomever you talk to at Small Dog, from our sales engineers to our shipping department, you are talking to a trained, enthusiastic, Apple Macintosh Specialist!

Small Dog Electronics is also an Authorized Apple Service Provider Plus. We have certified Apple Technicians that utilize genuine Apple parts for all repairs and provide technical support for our customers based upon their experience in upgrading and supporting thousands of Apple Macintosh computers for our customers!

Talk To an Apple Professional!

Did you know that when you call Small Dog Electronics, you are most likely talking to an Apple Product Professional? Each year, we participate in Apple's on-line courses and seminar training programs to learn as much as we can about the products that we sell and service!

Exclusive Top Dog Membership Treats

For each dollar purchased on-line at the Small Dog web site, you will receive a doggie treat. You can redeem your accumulated treats for Small Dog or Apple apparel and other products. You are automatically enrolled and begin accumulating treats in the Top Dog Club with your first purchase. Remember the more you buy, the more treats for you!



**Small Dog
Electronics**

1673 Main Street
Waitsfield, VT 05673 USA
Phone: 802-496-7171
Online: smalldog.com



Apple Specialist

example, a **Run** or **Reopen** handler in an AppleScript Studio script will never be called, even though it will compile because these events are proper AppleScript syntax: there is no **Run** or **Reopen** event in AppleScript Studio, so we can't connect them.

Event handlers are connected using Interface Builder. Before switching to Interface Builder to do this, however, we must first save our project in Project Builder. Due to the way the first release of AppleScript Studio is set up, there is a risk that we will lose most or all of our scripts if the aren't saved before switching to Interface Builder and connecting new event handlers. Therefore, compile and save the project now.

The first event handler we will connect is **Choose Menu Item** in **MainMenu.nib**. The user of Doyle must have some means of opening our Preferences window in order to review and set preferences, and in Mac OS X this is normally done by choosing the **Preferences** menu item in the Application menu. In Interface Builder, open **MainMenu.nib** and, if necessary, double-click the **MainMenu** icon in the Instances pane to open the menu bar. Click the Application menu on the left to open it, and see that it contains a Preferences menu item. If we were to compile and run the application now, we would find that this menu item is disabled. In order to enable it, we will have to connect an event handler to it. Click on the Preferences menu item to select it, then, in the AppleScript pane of the **NSMenuItem** Info panel, click **Choose Menu Item** under the Menu group in the Event Handlers area, click **PreferencesController.applescript** in the Script area at the bottom, and click the Edit Script button. A new **Choose Menu Item** event handler appears in **PreferencesController.applescript** in Project Builder. Fill in the new event handler with the statements shown in Listing 5 (the ellipsis, or three dots, are typed by pressing Option-;).

Listing 5

```
on choose menu item theMenuItem
    if title of theMenuItem is "Preferences..." then
        load nib "Preferences"
    end if
end choose menu item
```

Although not necessary at this point, we enclose the **Load Nib** command in an **If** block checking the name of the menu item. This will make it easier to connect other **Choose Menu Item** event handlers in this script in the future, if desired. AppleScript does not allow overloading of handlers, so scripts cannot contain multiple handlers having the same name, even if the direct parameters differ. Being limited to a single **Choose Menu Item** event handler in any one script, we have to test the parameter in chained **If/Else** clauses to find the one menu item that the user chose.

This event handler causes the **Preferences.nib** file to be loaded (notice that the ".nib" file extension is omitted), which will cause the window that we designed in Interface Builder to open. However, it will open invisibly, so we will have to tell it to become visible after the nib file has been loaded. Furthermore, we should set the visual state of its user controls

to match the state of the preferences file before making it visible, in order to avoid unsightly flashing. AppleScript Studio invokes a **Will Open** event handler whenever a window is about to open, so we will connect that event handler next.

Before switching to Interface Builder, save the project file to ensure that our scripts will not be lost. In Interface Builder, open **Preferences.nib** and, if necessary, double-click the Window icon in the Instances pane to open the Preferences window. Click in an empty area of the Preferences window to select it if something else is currently selected. Then, in the AppleScript pane of the **NSWindow** Info panel, click **Will Open** under the Window group in the Event Handlers area, click **PreferencesController.applescript** in the Script area at the bottom, and click the Edit Script button. A new **Will Open** event handler appears in **PreferencesController.applescript**. Fill in the new event handler with the statements shown in Listing 6.

Listing 6

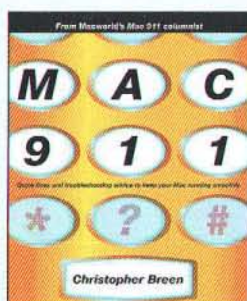
```
on will open theWindow
    if title of theWindow is "Preferences" then
        set state of button 1 of theWindow to
            to preferencesControllerLib's commentsAreShown()
        set visible of theWindow to true
    end if
end will open
```

This is the first time we have called a handler that resides in a script object. Here, we do so by qualifying the call with the possessive form of the script object's name, **preferencesControllerLib's**; we could as well have used AppleScript's alternative syntax, of **preferencesControllerLib**, or a **tell preferencesControllerLib** block. The single checkbox in the Preferences window is button 1 of that window (in Cocoa, a checkbox is implemented as a form of button), and we want to set its **State** property to checked or unchecked, according to the current value of the preference setting. The preference setting is saved as a **true** or **false** value in AppleScript terms. Although the Cocoa documentation tells us that a checkbox button's **State** value is an integer, capable of taking any of the values 1, 0, or -1, Cocoa allows 1 and 0 to be represented, respectively, as **YES** or **NO** (**true** or **false**). (In Cocoa, -1, or the constant **NSMixedState**, represents a mixed-state checkbox, portrayed with a dash in the checkbox instead of a checkmark.) We therefore get the return value of the **commentsAreShown()** handler in the **preferencesControllerLib** script object and set the **State** of the checkbox to the returned Boolean value. Since the script object is both declared within and called from **PreferencesController.applescript**, it isn't necessary to use the **Load Script** command to load it.

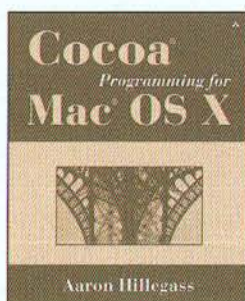
We know, from the work we have done so far, that invoking the **commentsAreShown()** handler causes **PreferencesController.applescript** to load the **preferencesModelLib** script object in **PreferencesModel.applescript** into the **PrefsModelLib** property, if it isn't already loaded, and to tell the **PrefsModelLib** property to execute its **GetShowCommentsPreference()** primitive handler, and that the latter in turn reads the preferences file from disk, pulls out the value of the desired preference item, and returns

Tips, Tricks, and Sage Advice for Your Mac

30% off*
these Mac
titles at
BORDERS®



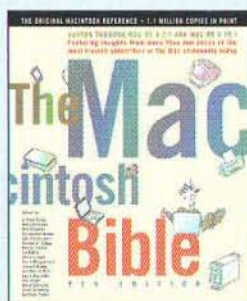
Mac 911
Chris Breen
352 pages • 0-201-77339-2
Retail Price: \$29.99
Borders Price: \$20.99



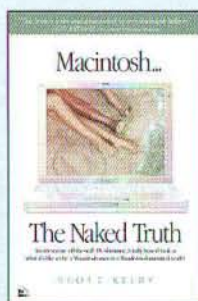
Cocoa Programming for Mac OS X
Aaron Hillegass
416 pages • 0-201-72683-1
Retail Price: \$44.99
Borders Price: \$31.49



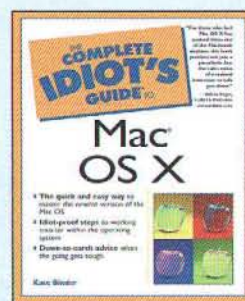
Mac OS X Advanced: Visual QuickPro Guide
Maria Langer
312 pages • 0-201-74577-1
Retail Price: \$24.99
Borders Price: \$17.49



The Macintosh Bible, 8th Edition
Clifford Colby and Marty Cortinas, Eds.
984 pages • 0-201-70899-X
Retail Price: \$34.99
Borders Price: \$24.49



Macintosh...The Naked Truth
Scott Kelby
240 pages • 0-7357-1284-0
Retail Price: \$19.99
Borders Price: \$13.99



Complete Idiot's Guide to Mac OS X
Kate Binder
384 pages • 0-7897-2528-2
Retail Price: \$19.99
Borders Price: \$13.99

*30% off these select titles 3/24/02 through 4/28/02.



BORDERS®

BOOKS • MUSIC • MOVIES • CAFE

it. All of this happens in a trice. Then, when the **Visible** property of the window is set to true, the window appears on screen, with a checkmark in the checkbox if the preference value on disk was true.

The final event handler needed by `PreferencesController.applescript` is a **Clicked** handler to update the value of the preference on disk when the user clicks the checkbox. Cocoa automatically supplies the checkmark when the user clicks. With `Preferences.nib` still open in Interface Builder, click on the checkbox to select it, then go to the AppleScript pane of the NSButton Info panel. In the Action group, check the **Clicked** event handler, and check the `PreferencesController.applescript` script near the bottom, then click the **Edit Script** button. In Project Builder, fill in the new **Clicked** handler according to Listing 7.

Listing 7

```
on clicked theControl
    if title of theControl is ~
        "Show Comments in Usage Log" then
            if state of theControl is 1 then
                tell preferencesControllerLib to showComments()
            else
                tell preferencesControllerLib to hideComments()
            end if
        end if
    end clicked
```

By now, you can trace out for yourself exactly how this works. We assume that, if the state of the checkbox is not 1 (**true**), then it must be 0 (**false**), because we do not make use of the available third state of a Cocoa checkbox in Doyle.

There is still one important thing missing from our preferences system: the preference file doesn't get initialized to default values if no preferences file is found when the application is launched. We want our preferences to be initialized as soon as the application has finished launching, so an appropriate event handler to connect is the **Will Finish Launching** event handler in the Application Suite. This is equivalent to Cocoa's `awakeFromNib` method. It is invoked by the system after the application's nib files have been loaded, the user interface has been initialized, and other Cocoa initialization has taken place.

The `MainMenu.nib` file is the main nib file for the application, and the File's Owner of that nib file is Cocoa's `NSApplication` class, so this is where to connect the **Will Finish Launching** event handler. In Interface Builder, open `MainMenu.nib`, click the File's Owner icon in the Instances pane to select it, and open the AppleScript pane of the File's Owner Info panel. Expand the Application group and click to select the checkbox beside the **Will Finish Launching** event handler. Then click to select `Application.applescript` at the bottom of the File's Owner Info panel, and click the **Edit Script** button.

When you switch back to Project Builder, you will find that a **Will Finish Launching** handler has been added to `Application.applescript`. The `Application.applescript` script was supplied by the AppleScript Document-based Application template when we first created the project, but it was empty. Type to fill in the statements in the new **Will Finish Launching** handler as shown in Listing 8.

Listing 8: Application.applescript

```
(* Application.applescript *)

(* Properties *)

property prefsControllerLib : null

(* Handlers *)

on loadPreferencesControllerLib()
    -- Loads PreferencesControllerLib from PreferencesController.applescript, if not loaded.
    if class of prefsControllerLib is not script then
        load script POSIX file ~
            ((path for script "PreferencesController" ~
                extension "scpt") of main bundle)
        set prefsControllerLib to ~
            preferencesControllerLib of result
    end if
end loadPreferencesControllerLib

(* Event handlers *)

on will finish launching theObject
    -- Initializes the application after main nib files are loaded and initialized by Cocoa.
    -- Connected to File's Owner in MainMenu.nib

    -- Initialize preferences to default values, if preferences file not found
    loadPreferencesControllerLib()
    tell prefsControllerLib to initPrefs()
end will finish launching
```

You know from earlier work what chain of events this event handler unleashes.

Notice, however, that this is the first time we have called a method in `PreferencesController.applescript` from outside of that script, in this case, from `Application.applescript`. We therefore needed a handler to load the `preferencesControllerLib` script object from `PreferencesController.applescript` into a property in `Application.applescript`. We did this the same way we loaded a script property from `PreferencesModel.applescript` into `PreferencesController.applescript`. The necessary property and handler are shown in Listing 8.

Until Next Time

This installment is now complete. We have a working preferences system for Doyle.

To check it out, first compile Doyle for deployment. In Project Builder, open the Targets pane by clicking the Targets tab. In the Build Styles area at the bottom, select the Deployment radio button, then rebuild the application. In the Finder, open the Build subfolder in the Doyle project folder, and drag a copy of the Doyle application into your Applications folder. Double-click it to launch Doyle and see its main window open. Then choose **Doyle -> Preferences** and see the Preferences window open. If this is the first time you have run Doyle, there was no Doyle Preferences file in existence, so you should see that the button is checked to reflect the default value of the preference. Uncheck it and close the Preferences window. Reopen the Preferences window, and you will again see it unchecked, proving that it successfully wrote the new preference value to disk and retrieved it. You can even quit and relaunch Doyle and perform the test again, if you don't believe it.

In the next installment, we will write the necessary scripts to make the Doyle Usage Log work, and to save and retrieve its values in an AppleScript Studio document. We will, of course, use our new preferences system to determine whether the Comments column in the Log window should be shown.

**AS A
PROGRAMMER
OR SYSADMIN,
YOU'VE GOT
BETTER THINGS
TO DO THAN
FIGHT WITH A
WEB SITE**



It's time for you to take a look at MGI

MGI, a plug-in to 4D's award-winning server, WebSTAR, is designed to provide functionality to otherwise static web sites, whether for a private intranet or enterprise - class ISP/ASP. MGI was specifically developed to be used by web graphic designers with no scripting or programming experience. If you know HTML, you know MGI. And if you are a programmer, you can learn MGI by the time your pizza is delivered. You can try out MGI for free - right now - without obligation by downloading a fully - functioning demo at :

<http://www.pageplanetsoftware.com>



SOFTWARE BUILT WITH YOU IN MIND

Searchable Databases
Online Stores
Info Baskets
Guestbooks
Banner Ads
Credit Card Transactions
Counters
Password Protection
Surveys
Quizzes
Form Processing
Conditionals
Math
Cookies

Date and Time
File Includes
File Writes
Web Based Email
Classified Ads
Discussion Groups
Web Chat
Affiliate Tracking
I/O Transactions
Δ Time Calculations
PGP Encryption
Token Tracking
Calendars
Random Rotation
Send Mail
E - Cards
Credit Bank

by Tim Monroe

Broadcast News

Broadcasting Movies Over a Network

INTRODUCTION

The key new technology introduced in QuickTime 4 was support for receiving real-time streamed data. That is to say, QuickTime-savvy applications like QuickTime Player or our own QTShell-based sample applications can receive video, audio, and other kinds of data streamed across the Internet. Real-time streaming, unlike the progressive downloading of movie files that has been available since QuickTime 3, can handle live data and does not require downloading potentially huge files onto the user's computer; this permits QuickTime playback applications to support uses such as video-on-demand and rebroadcast streaming.

The real-time streaming provided by QuickTime 4 was a client-side technology only; it did not provide any means to serve up, or transmit, the data streams. At that time, special software was required to create the streams of data that could be sent out over a network and then received and played back by QuickTime-savvy applications. As we'll see, the protocols used in transmitting the data from the server to the client conform to Internet Engineering Task Force (IETF) standards, but implementing those protocols in a transmitter application required intimate knowledge of a handful of IETF specifications, as well as a good bit of programming.

QuickTime 5 provides a set of broadcasting functions that allow us to create transmitter applications. For example, we can take the audiovisual data captured by a camcorder attached to one computer and broadcast that data to other computers on a network. Together, the transmitter technologies provided by QuickTime 5 and the receiver technologies provided by QuickTime 4 give us the complete set of tools we need to send audiovisual streams from one computer and view them on another. The good news here is that we need to know virtually nothing about the applicable IETF specifications to do all this; the really good news is that the amount of code we need to write to create a broadcasting

application is surprisingly small. Indeed, we'll be able to write this application using fewer than a dozen of these new broadcasting functions.

In this article, we're going to see how to use these new APIs to build a sample application that broadcasts live data to other computers. (Let's call this application QTBroadcast.) In a previous *QuickTime Toolkit* article ("Captured", in *MacTech* December 2001), we saw how to use the sequence grabber to capture sound and video data from devices attached to our local computer; here we'll see how to send captured data to other computers located remotely on a network.

When it starts up, QTBroadcast automatically opens a *monitor window*, shown in **Figure 1**; this is modeled on the monitor window we created in the previous article, but now contains a button to start and stop the broadcasting.



Figure 1: The monitor window of QTBroadcast

The Test menu of QTBroadcast is shown in **Figure 2**; as you can see, it contains items that allow the user to select an SDP file (which we'll describe later) and to configure the broadcasting settings.

Tim Monroe is a member of the QuickTime engineering team at Apple. You can contact him at monroe@apple.com.

Test

Select SDP File...	⌘1
Configure Settings...	⌘2

Figure 2: The Test menu of QTBroadcast

We'll begin by surveying the various streaming capabilities provided by QuickTime, and we'll take a brief look at the protocols that underlie those capabilities. Then we'll consider the code we need to add to our basic application shell to allow it to broadcast live data streams. Note that QuickTime's broadcasting APIs are currently available only on Macintosh operating systems (Mac OS 8 and 9, and Mac OS X).

QUICKTIME STREAMING

Streaming is the process whereby one computer (the *transmitter*) chops a file or sequence of bytes up into discrete chunks (called *packets*) and sends them across a network to another computer (the *receiver* or *client*). The client's job is to reassemble the packets and do the right thing with them. The series of packets is a *stream*. For present purposes, we are interested only in streams of audiovisual data that can be reassembled and played back as a QuickTime movie.

It's important to keep in mind that QuickTime has supported a kind of streaming, called *HTTP streaming* or *progressive downloading*, ever since QuickTime 3 was released in 1998. QuickTime's HTTP streaming allows the QuickTime browser plug-in to begin playing a movie embedded in a web page before the entire movie has downloaded to the local computer. HTTP streaming is essentially a file transfer protocol, in the sense that the entire movie is downloaded to the user's computer. One key advantage of QuickTime's HTTP streaming is that a user can begin interacting with web-based content before the entire movie arrives on the user's computer. As we saw in an earlier article, the movie file embedded on the web page is a standard QuickTime movie file saved in Fast Start format (where the movie atom is the first atom in the movie file).

QuickTime 4 introduced a different kind of streaming, called *real-time streaming*. With real-time streaming, packets are sent out over the network in real time (so that, for instance, a 10-minute movie would take 10 minutes to download). When packets are received by the client application (for instance, by QuickTime Player), they are reassembled into a QuickTime movie and played for the user. In general, packets are discarded as soon as they have been played, so no file is ever created on the user's local storage devices.

QuickTime's real-time streaming uses an open IETF streaming protocol known as the *Realtime Transport Protocol* (RTP) to send the packets of video and audio data. For control information, such as establishing a connection between the client and server or telling the server to jump to a new time in a video-on-demand movie, QuickTime uses a different protocol known as the *Realtime Streaming Protocol* (RTSP). **Figure 3**



Results of

Clean Code

```
if ("you_need_training") {  
    if ("you_need_resources") {  
        if ("you_need_sanity") {  
            if ("you_need_it_now") {  
                goTo("mediaschool");  
            }  
        }  
    }  
}
```



Mac OS X Developer Training

<http://www.mediaschool.com/adc/>

shows the basic network protocols used by QuickTime's streaming architecture. RTSP uses the TCP/IP transport layer, while RTP uses the lower-level UDP/IP transport layer.

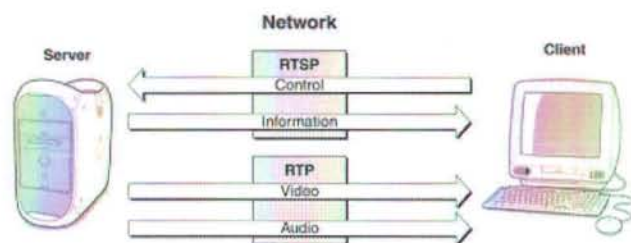


Figure 3: The basic QuickTime streaming protocols

Because RTP uses UDP/IP, it does not guarantee delivery of packets. So it's possible for some packets to get lost in transit and never arrive at the client computer. This packet loss can result in some degradation of the video and sound when the movie is played back on the client computer. If you need guaranteed delivery of packets, you should use HTTP streaming, which uses TCP/IP to ensure that all transmitted packets are in fact received (by retransmitting any lost packets).

QuickTime supports streaming of video, audio, and text data in any of the formats that can be played locally by QuickTime, including AVI, Sorenson, QDesign, MP3, and MIDI. It cannot currently stream movies that contain sprite tracks or that incorporate features that depend on track references, such as QuickTime video effects, chapter lists, and some tweening effects. To handle these sorts of movies, you can either use HTTP streaming or you can create movies that store some data locally and combine that data with data received via RTP streaming. You can even create movies some of whose tracks are delivered via HTTP streaming and some via RTP streaming.

The files that reside on the streaming server are standard QuickTime movie files, with one important addition: each track in the movie that is to be streamed across the network must be accompanied by a *hint track*. The hint track contains information that tells the server software how to packetize the corresponding streamed track. In other words, the hint track is a sort of blueprint for creating streams of data. Without the hint track, the server software would have to know a great deal about the particular audio or video format contained in the streamed track, so that it would know how best to chop the data up into packets (for instance, possibly duplicating some of the packets to protect against losing important frames of video). The hint track insulates the server software from having to know anything about the actual video or audio data it's serving.

This is especially important because it allows streaming servers to run on operating systems that don't even run the QuickTime client software. Indeed, the first commercially available QuickTime server application, Apple's QuickTime Streaming Server (QTSS), debuted on machines running the Mac OS X Server software. Moreover, Apple has released the source

code for the QuickTime Streaming Server under the Apple Public Source License, which conforms to the Open Source community guidelines. Versions of QTSS currently run under Windows and various flavors of UNIX, including Linux.

QUICKTIME BROADCASTING

The broadcasting APIs introduced in QuickTime 5 allow us to create broadcasting applications (or, more simply, broadcasters). A *broadcaster* is an application that takes data from a source other than a hinted movie, compresses that data (if necessary), packetizes that data into streams, and then sends the streams out over a network. In the simplest case, the streams are targeted at a single specific client on the network. This type of serving is called *unicasting*. It's also possible to target a set of streams at more than one client on the network, using a special reserved address; this type of serving is called *multicasting*.

The QuickTime broadcasting APIs support both unicasting and multicasting. Some routers, however, are not configured to allow multicast streams. In this case, we still need to use a streaming server (such as QTSS). We can transmit the streams to a machine running QTSS, which then unicasts distinct streams to any number of remote clients.

The broadcasting APIs can be divided into two general categories, which we'll call the *sourcer* APIs and the *presentation* APIs. We can use the *sourcer* APIs to select and configure a source for the broadcast data. (A *sourcer component*, or *sourcer*, is a component that can read data from a specific kind of source.) Currently QuickTime supports broadcasting data from any of these kinds of sources:

- Data captured using a sequence grabber component from audiovisual devices
- Movie files stored locally on the transmitter
- Precompressed media data that is not stored in a movie file

This last kind of source allows us to broadcast virtually any kind of media data; we simply need to pass the appropriate data, a sample description for that data, and (optionally) a timestamp and duration. Keep in mind, however, that the client receiving the broadcast needs to have a media handler for that kind of data. Also, some kinds of data (for instance, sprite media data or Flash media data) do not respond well to packet loss and hence are currently not good candidates for broadcast streaming.

We use the *presentation* APIs to present (or broadcast) the data provided by a sourcer. A *presentation* is a collection of one or more streams of data, which can consist of packets of audio, video, text, or other data. You might think of a presentation as the streaming equivalent of a movie and the streams within the presentation as the streaming equivalent of a movie's tracks. The client application receives the presentation and reassembles it into a movie, which it plays back in exactly the same way it plays a local movie.

The sourcer and presentation APIs are a marvel of simplicity. As I mentioned earlier, we will be able to develop a broadcasting application using fewer than a dozen broadcasting

functions. In fact, for the moment we won't need to use any of the sourcer functions at all, since we'll rely on QuickTime's default behavior of using the sequence grabber as the source for the broadcast data. So let's get started.

Setting Up For Broadcasting

Before we can begin broadcasting live audio and video data captured from an audiovisual device attached to our computer, we need to initialize our application for broadcasting and then create a presentation. As we've seen, a presentation is analogous to a movie: it consists of one or more streams of data targeted at one or more remote computers. In QTBroadcast, we shall allow at most one active presentation at a time, and the data being broadcast will be displayed in the monitor window (see Figure 1 again), which also contains a button that can be used to start and pause the broadcast. To keep track of the presentation and the monitor window, we'll use these global variables:

```
QTSPresentation      gPresentation =
kQTSInvalidPresentation:
QTSTNotificationUPP  gNotificationUPP = NULL;
DialogPtr            gMonitor = NULL;
UserItemUPP          gMonitorUserItemProcUPP = NULL;
Boolean              gBroadcasting = false;
```

The `gPresentation` global variable is an identifier for the single presentation supported by QTBroadcast; as you can see, it's of type `QTSPresentation` and is initialized to the value `kQTSInvalidPresentation`. Associated with the presentation is a *presentation notification procedure*, which we identify using the `gNotificationUPP` global variable. Our notification procedure is called on specific events involving the presentation, such as when a presentation is first created and when a connection to the client machine occurs.

We use the `gMonitor` and `gMonitorUserItemProcUPP` global variables to keep track of the monitor window and the user item in that window (where we draw the video data that's being broadcast). Finally, we'll use the `gBroadcasting` global variable to keep track of whether we're currently broadcasting data or not.

When QTBroadcast starts up, it calls the `QTBC_Init` function, defined in Listing 1, to create the monitor window and to allocate the universal procedure pointers `gNotificationUPP` and `gMonitorUserItemProcUPP`.

Listing 1: Initializing for broadcasting

```
OSErr QTBC_Init (void)                                     QTBC_Init
{
    OSErr      myErr = noErr;

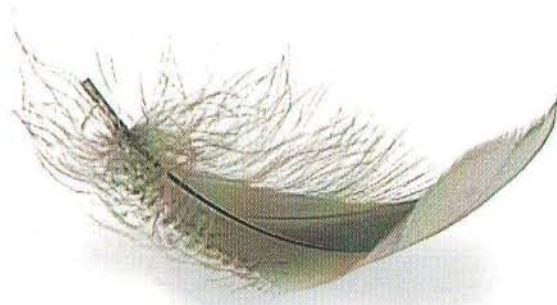
    // allocate global storage
    gNotificationUPP = (QTSTNotificationUPP)
        NewQTSTNotificationUPP(QTBC_NotificationProc);
    if (gNotificationUPP == NULL) {
        myErr = paramErr;
        goto bail;
    }

    gMonitorUserItemProcUPP = NewUserItemUPP
        (QTBC_UserItemProcedure);
    if (gMonitorUserItemProcUPP == NULL) {
        myErr = paramErr;
        goto bail;
    }
}
```



Results of

Sloppy Code



We can help clean up your act.



Mac OS X Developer Training

<http://www.mediaschool.com/ad/>


```

}

// open the monitor window
gMonitor = QTBC_CreateMonitorWindow();
if (gMonitor == NULL) {
    myErr = memFullErr;
    goto bail;
}

bail:
// if an error occurred, clean up
if (myErr != noErr)
    QTBC_Stop();

return(myErr);
}

```

QTBC_Init calls NewQTSNotificationUPP and NewUserItemUPP to create the two UPPs, and it calls the application function QTBC_CreateMonitorWindow to create and display the monitor window. We'll consider QTBC_CreateMonitorWindow in depth a bit later (Listing 9); for the moment, it's sufficient to know that it calls GetNewDialog to create a dialog window from information in the application's resources.

If any error occurs in QTBC_Init, we don't want to continue. In that case, we'll call the QTBC_Stop function, which is defined in Listing 2. QTBC_Stop simply deallocates the two UPPs created by QTBC_Init and disposes of the monitor window.

Listing 2: Shutting down broadcasting

QTBC_Stop

```

void QTBC_Stop (void)
{
    // deallocate any global storage
    if (gNotificationUPP != NULL) {
        DisposeQTSNotificationUPP(gNotificationUPP);
        gNotificationUPP = NULL;
    }
    if (gMonitorUserItemProcUPP != NULL) {
        DisposeUserItemUPP(gMonitorUserItemProcUPP);
        gMonitorUserItemProcUPP = NULL;
    }

    // close the monitor window
    if (gMonitor != NULL)
        DisposeDialog(gMonitor);
}

```

We also call QTBC_Stop just before QTBroadcast terminates; at that time we will also call the QTBC_StopBroadcasting function, which calls QTSDisposePresentation to dispose of the presentation gPresentation. (QTBC_StopBroadcasting is defined later, in Listing 7.) It's important to dispose of the UPPs and the monitor window only *after* we dispose of the presentation itself; accordingly, our application-specific shutdown routine, QTApp_Stop, contains these two lines of code:

```

QTBC_StopBroadcasting();
QTBC_Stop();

```



THE LAW OFFICE OF
BRADLEY M. SNIDERMAN

Need help safeguarding your software?

If you're developing software, you need your valuable work protected with trademark and copyright registration, as well as Non Disclosure Agreements.

Then, when you are ready to sell it, you can protect yourself further with a licensing agreement.

I am an attorney practicing in Intellectual Property, Business Formations, Corporate, Commercial and Contract law.

Please give me a call or an e-mail. Reasonable fees.

23679 Calabasas Rd. #558 • Calabasas, CA 91302

PHONE 818-222-0365 FAX 818-591-1038 EMAIL brad@sniderman.com

Your Data Isn't an Important Part of the Job — It *IS* the Job.

VXA® FireWire

The High-Performance Tape Storage Solution For Apple Users

*"I can stick
my entire
hard drive
onto a tape
7 times
—that's just
plain cool!"*

Other backup media, such as CD-RWs and DVD-RAMs, simply can't compare to VXA-I tape technology when it comes to capacity, transfer rate and overall price.

Fast, economical and easy-to-use, VXA FireWire offers:

Ultimate File Security

- 100% Data Restore and Interchange
- Plug-and-Play Connectivity

Sharable Media

- Multi-Gigabyte File Transport
- Portable, Hot-Pluggable

Real Time Digital Video

Cross-Platform Compatibility

- FireWire/IEEE1394/iLink Supported By All Major Manufacturers



Technology	Capacity in MBs	Transfer Rate in MB/sec	Price per MB
VXA Tape	33000	3	\$0.03
Imation Travan	10000	1	\$0.05
DVD RAM	9400	2.7	\$0.05
CD RW	700	1.9	\$0.50
Zip	250	1.2	\$0.76

LOWEST
COST per MB
Of ANY
Technology

Call Exabyte sales at 1-800-774-7172
or visit us on the web at www.exabyte.com



Tape Storage By
Exabyte®

Creating a Presentation

If QTBC_Init completes successfully, the monitor window is displayed on the screen but it contains no image and the Start button is inactive (as shown in **Figure 4**).

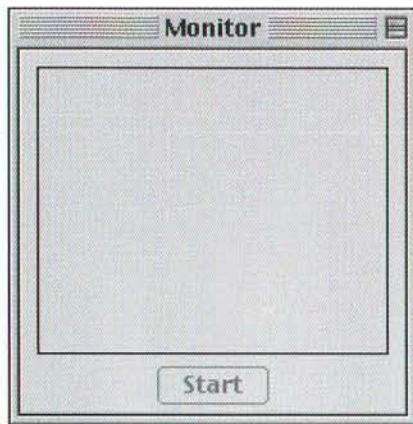


Figure 4: The monitor window on application launch

Before we can begin broadcasting data, we need to create a presentation. We do this by calling the `QTSNewPresentation` function, like this:

```
myErr = QTSNewPresentation(myPresParamsPtr, &gPresentation);
```

The second parameter is the address of our global variable `gPresentation`; if `QTSNewPresentation` completes successfully, it returns an identifier for the new presentation in that variable. The first parameter is the address of a *new presentation parameters structure*, which is defined like this:

```
struct QTSNewPresentationParams {
    OSType          dataType;
    const void *    data;
    UInt32          dataLength;
    QTSEditListHandle editList;
    SInt32          flags;
    TimeScale       timeScale;
    QTSMediaParams * mediaParams;
    QTSNotificationUPP notificationProc;
    void *          notificationRefCon;
};
```

So before we can call `QTSNewPresentation`, we need to allocate a new presentation parameters structure and fill in most of its fields. We allocate this structure by calling `NewPtrClear`:

```
myPresParamsPtr = (QTSNewPresentationParams *)
    NewPtrClear(sizeof(QTSNewPresentationParams));
```

The first three fields of the new presentation parameters structure contain information about some of the network settings to be used by the presentation, such as the destination IP address for the broadcast and the ports to use for the data streams. The IETF has defined a standard format for this information, called the *session description protocol* (SDP). An *SDP file* (that is, a file that conforms to the SDP) is used by clients and servers for initiating a network connection and transfer of multimedia streams between a server and

its clients. This file describes the types and formats of the media to be transferred, the transport protocols, and the addresses to which the media are to be streamed. Here's a sample SDP file:

```
v=0
c=IN IP4 224.2.1.2/15/1
m=audio 1000 RTP/AVP 12
m=video 2000 RTP/AVP 101
a=rtpmap:101 H263-1998
```

The line beginning with "v=" specifies the protocol version, which currently is 0. The line beginning with "c=" specifies the connection information, which consists of a network type and address type (here, "IN IP4" for IP version 4 addressing on the Internet), the destination address (here, "224.2.1.2", which is an address reserved for multicasting), a time-to-live value (here, 15), and the number of contiguous multicast addresses (here, 1). The lines beginning with "m=" indicate which transport protocols and ports to use for specific media types. In the file shown above, the audio data is to be sent via RTP to port 1000 and the video data is to be sent via RTP to port 2000. Finally, the line beginning with "a=" specifies media attributes. In this case, the `rtpmap` attribute provides information about dynamic payload binding.

In our sample application `QTBroadcast`, the user selects an SDP file using the first menu item in the Test menu (namely, "Select SDP File..."). In response to menu item, we call the `QTBC_SetupPresentation` function, which in turn uses our standard framework function `QTFrame_GetOneFileWithPreview` to allow the user to select an SDP file. `QTFrame_GetOneFileWithPreview` returns a file system specification for that file. We can then fill in the relevant fields of the new presentation parameters structure like this:

```
myPresParamsPtr->dataType = kQTSSFileDataType;
myPresParamsPtr->data = &myFSSpec;
myPresParamsPtr->dataLength = sizeof(myFSSpec);
```

QuickTime also supports SDP information stored in memory (in which case the `dataType` field should be set to `kQTSSDPDataType`). In `QTBroadcast`, we'll restrict our attention to SDP files only.

The `editList` field of the new presentation parameters structure is a handle to a `QTSEditList` structure, which holds information about edits for stored streams. For live broadcasting, we can ignore that field.

The `flags` field of the new presentation parameters structure contains information for the presentation we are about to create. Currently these flags are defined:

```
enum {
    kQTSAutoModeFlag          = 0x00000001,
    kQTSDontShowStatusFlag    = 0x00000008,
    kQTSSendMediaFlag         = 0x00010000,
    kQTSReceiveMediaFlag      = 0x00020000
};
```

The `kQTSAutoModeFlag` flag indicates that the presentation should be automatically configured to its default settings; most

HASP[®]

One Key, Multiple Platforms

www.eAladdin.com

Now your cross-platform software enjoys cross-platform security. By preventing unauthorized software usage and duplication, HASP protects intellectual property rights and increases revenues across the board. A single HASP USB protects Windows, Linux and Mac applications, for reduced development time, simplified shipping logistics and lowered costs. Our cross-platform software protection key benefits your customers too – now they can use your application on multiple platforms with maximum convenience. More developers than ever rely on HASP to increase revenues.

HASP supports:

- Windows 3.x/95/98/ME/2000/NT4/XP • Linux
- Mac OS 8.6-9.x • Mac OS X
- TCP/IP, IPX, NetBIOS-based networks



To order your HASP Developer's Kit, visit:
www.eAladdin.com/mactech
or call 800-223-4277 EST
& 800-562-2543 PST,CST,MST

importantly, this means that the default sourcer is to be used, which is the sequence grabber. The `kQTSDontShowStatusFlag` flag indicates that we do not want QuickTime to create a streaming status handler for the presentation; if we request a status handler, it displays connection and status information in the monitor window (that is, in the window associated with the sequence grabber). For instance, when we first create a presentation, QuickTime attempts to connect to the specified target or targets and displays the status message shown in **Figure 5**.

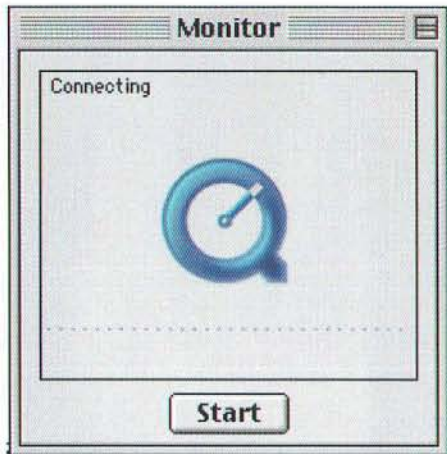


Figure 5: The status message while connecting

When the presentation is ready and the requisite network connections have been established, we'll see the message shown in **Figure 6**. (Keep in mind that these status messages appear on the transmitter only; the broadcasting client also displays some status information while establishing a connection to the transmitter, which we'll encounter later.)



Figure 6: The status message when ready

The remaining two flags, `kQTSSendMediaFlag` and `kQTSReceiveMediaFlag`, indicate whether we want to send or receive data with the presentation we're creating. In QTBroadcast, we set our flags like this:

```
myPresParamsPtr->flags = kQTSAutoModeFlag |
                        kQTSDontShowStatusFlag | kQTSSendMediaFlag
```

The `timeScale` field of the new presentation parameters structure is currently unused (as far as I can determine); in QTBroadcast, we set that field to 0.

The `mediaParams` field holds a pointer to a *media parameters structure*, of type `QTSMediaParams`:

```
struct QTSMediaParams {
    QTSVideoParams    v;
    QTSAudioParams    a;
};
```

As you can see, this structure contains a *video parameters structure* and an *audio parameters structure* (of types `QTSVideoParams` and `QTSAudioParams` respectively); these structures contain information about the video and audio media data being previewed locally — that is, the data being displayed in the monitor window and the sound played through the computer's speakers. In QTBroadcast we want to leave the audio settings at their default values, but we want to configure the video settings so that the preview draws in the correct location in the monitor window. We begin by allocating a new media parameters structure:

```
myMediaParamsPtr = (QTSMediaParams *)
    NewPtrClear(sizeof(QTSMediaParams));
myPresParamsPtr->mediaParams = myMediaParamsPtr;
```

At this point we want to call the `QTSInitializeMediaParams` function, which initializes the media parameters structure to some default values:

```
myErr = QTSInitializeMediaParams(myMediaParamsPtr);
```

Now let's configure the video previewing. The video parameters structure looks like this:

```
struct QTSVideoParams {
    Fixed          width;
    Fixed          height;
    MatrixRecord   matrix;
    CGrafPtr       gWorld;
    GDHandle       gdHandle;
    RgnHandle      clip;
    short          graphicsMode;
    RGBColor       opColor;
};
```

Notice (in Figure 1, once again) that the preview images are drawn in the user item rectangle, which is 176 pixels wide and 144 pixels high and is offset from the window origin by 10 pixels vertically and horizontally. So we'll use these lines of code to place the images in the proper location:

```
myPresParamsPtr->mediaParams->v.width = Long2Fix(176);
myPresParamsPtr->mediaParams->v.height = Long2Fix(144);

TranslateMatrix(&(myPresParamsPtr->mediaParams->v.matrix),
    Long2Fix(10), Long2Fix(10));
```

And we'll set the port and graphics device like this:

```
myPresParamsPtr->mediaParams->v.gWorld =
    GetDialogPort(gMonitor);
myPresParamsPtr->mediaParams->v.gdHandle = NULL;
```


personals

like ships
passing in
the night

SUPERMARTIN I'm not older
gal with Spidey working
at WFM. Me: 23, open to
pony tail. I would like to
#6922

COOL I did not
get made my day,
you beautiful. I would
in. Did I read

PHONE-PLAYING
Athville recently
Georgia. #6854

WE MET IN MADISON,
summer. Would like to talk
again! #6967

FROM WCW. Exchanged
ces at cage match. It was pure
agic. Would love to get you in a
eper hold. #5627

BEAUTIFUL AND SEVENTEEN:
Met you at the Metro. You were on a
date with someone else. Next time it
will be me. #6973

LEVITATING BUDDHA SWORD-
PLAY wedding lady. I'm interested
but your brave new world inspired
with you. #6788

MANIC GOLF DANCER

VEGETARIAN BOWLER. You
bought me a warm beer and stole
my heart. Used same kind of ball
and spoke of hatred of rented
shoes. Would love to chat over
hummus. #5684

LAWN CARE? My husband got lazy
and hired you to mow our lawn.
Instead you landscaped my erotic
fantasies in ways I have never imag-
ined. Could not pronounce your
name but looked very sensual. I had
blue shoes on. #3696

TWINS WHO SAW TWINS. Us: two
handsome guys in suspenders walk-
ing Maltese. You: two foxy ladies
fighting over last piece of gum. What
do you say the four of us make two
good looking couples? Twin love.
Call me. Call me. #4747

DUGOUT FIRECRACKER. You
were cleaning up a beer that you
spilled on your white t-shirt and
threw a whiskey bottle at the
umpire. Must meet you and make
children. #5551

LORIN, YOU'RE GORGEOUS,
funny and brilliant. I don't deserve
you but a girl can dream. #6885

SY FROM DOWN SOUTH. You sat
with us at Smitty's 11/24, missed
you at The Boot. Wanna meet after
work sometime? Call and gimme
your number Jerry. #8927

CLASSY LATINA With substance,
5'6" plus, non smoker, very pretty, spe-
cialized chair, intelligent, loves
Spanish, possibly gay, married,
#6788

RON FROM SANTA FE. You
danced with me at the Hattie &
Cattle Club. Thanks! I was shy. Can
I see you again? Will come down to
look for you Friday night. #6841

GORGEOUS, WITTY, BORN TO
tease: love theater, dance, golf,
warm conversation. If you're tall,
35-55, non-smoker, financially
secure, enjoys pampering a woman,
traveling, long walks and stars,
please call. #6821

ME: LONELY SWEDISH LINGERIE MODEL
and gourmet cook. You: slightly overweight
and without ambition. Must be into computers,
role-playing games and air hockey. #5988

49; PLAIN BUT WITH GOOD BITS.
overweight but curvy, great mind,
wicked sense of humor, and a weird
view of life looking for like minded
person. Age not important. #6994

TREE HUGGER, MID 50'S, light
smoker, tall. Like easy living, tropics
and I'm friendly. Seeking consider-
ate, semi-fit companion with a clue.
Must love dogs and reggae. #6963

ARE YOU HONEST, handsome,
successful, financially secure, intelli-
gent, world traveled, cultured, cre-
ative, fit, playful, adventurous, pas-
sionate, humorous, caring, loving,
and between 40 and 50? Reply to
European, blonde female coun-
tess. #5852

DWN 5' ATTRACTIVE

ARE YOU STIMULATED BY beauty,
intelligence, humor? Attractive SWF
wants good looking SWM or SHM
for romantic adventures, possible
long term. Essentials: honesty, pas-
sion, kindness, sensuality, integrity,
open mind. #6741

ATTRACTIVE TALL (5'10"), slender
OPWF: 46, emotionally and physi-
cally fit, youthful appearance and
outlook, intelligent, loving, desires
long term relationship. #6853

YOU WON'T BELIEVE YOUR EYES
when you see this very cute, petite,
DWF: 46, long brown hair/hazel
eyes, 5' 110, outgoing personality
seeks DWN, 46-55, non smoker, fit,
college educated. Call me, let's see
if the chemistry is right! #6951

SWF 26, STRAWBERRY CURLS
and below. Craving a sharper,
middle-aged, educated, intelligent
man who is a real man. #6788

NOT SO DESPERATELY seeking
one smart, strange, sexy boy to
court and spark. Me: 23, open to
possibilities and ravenous for new
life experiences. #6933

SWINGING SANTA. Lonely man
who only works 6 weeks a year
seeking woman with full time
employment with benefits looking to
grow old with man who shakes like
a bowl full of jelly. #1258

WM, 95, RECENTLY WIDOWED,
seeking 18-20 hottie for "fun". Call
soon. I'm not getting any younger.
I'll put you in my will. #6757

BALD ROMEO. You serenaded the
old people at the old people home
last weekend. You were a terrible
singer and quite unattractive, but
your heart is obviously pure gold.
My sister would be perfect for you.
#7867

MONKEY TRAINER. Seeking
woman to train my monkey.
Seriously, his name is Murphy and
he is a 3 year old chimpanzee. He
likes pop tarts and nice people.
Plus, you and I will have sex.
#7874

SINGLE MAN. Single man seeking
single woman for relationship. I
enjoy dating and talking on the
phone to women that I am dating.
Would love a chance to date some-
one. #1254

CUTIE PIE SMARTY PANTS 40-50
seeks a hot, intelligent, and
magnetic, confident, sexy, Mus-
lim, who can hold the real thing.

SIDESH
sidesh
well p
for r
ror
ar

F
v
c
h

T
h
ing
fight
do y
good
Call me.

MANY WON
my life - but no
wonderful woman,
smart, professional, etc.
(non smoker). Love of nature,
irreverent humor. #6772

RECENTLY PAROLED, looking for
a lady who will keep me on the
straight and narrow. Must be into
drugs and thumping. #6357

HOTELIER ROMANTIC, seeks
a lady who will keep me on the
straight and narrow. Must be into
drugs and thumping. #6357

We're Easier.

Create anything from prototypes to full professional applications. Just drag and drop interface elements while REALbasic handles the details. You concentrate on what makes your stuff great — your ideas! REALbasic creates native compiled applications for Macintosh, Mac OS X and Windows without platform-specific adjustments. It's the powerful, easy-to-use tool for creating your own software. Each version of your software looks and works just as it should in each environment.

Complex problems shouldn't require complex solutions. The answer is REALbasic.



Download a free demo. www.realbasic.com

Finally, the `notificationProc` and `notificationRefCon` fields of the new presentation parameters structure specify the notification procedure and a reference constant that is passed to the notification procedure.

```
myPresParamsPtr->notificationProc = gNotificationUPP;
myPresParamsPtr->notificationRefCon = 0L;
```

At long last, we are ready to call `QTSNewPresentation` and `QTSPresPreview`:

```
myErr = QTSNewPresentation(myPresParamsPtr, &gPresentation);
myErr = QTSPresPreview(gPresentation, kQTSAAllStreams, NULL,
    kQTSNormalForwardRate, 0);
```

Listing 3 gives the complete definition for `QTBC_SetupPresentation`.

Listing 3: Setting up a presentation

```
OSErr QTBC_SetupPresentation (void)
{
    QTSNewPresentationParams *myPresParamsPtr = NULL;
    QTSMediaParams *myMediaParamsPtr = NULL;
    FSSpec myFSSpec;
    OSType myTypeList[] = {kQTFileTypeText};
    short myNumTypes = 1;
    QTFrameFileFilterUPP myFileFilterUPP = NULL;
    OSErr myErr = noErr;

    #if TARGET_OS_MAC
        myNumTypes = 0;
    #endif

    // create a new presentation parameters structure
    myPresParamsPtr = (QTSNewPresentationParams *)
        NewPtrClear(sizeof(QTSNewPresentationParams));
    if (myPresParamsPtr == NULL) {
        myErr = MemError();
        goto bail;
    }

    // create a new media parameters structure
    myMediaParamsPtr = (QTSMediaParams *)
        NewPtrClear(sizeof(QTSMediaParams));
    if (myMediaParamsPtr == NULL) {
        myErr = MemError();
        goto bail;
    }

    // initialize the media parameters to default values
    myErr = QTSInitializeMediaParams(myMediaParamsPtr);
    if (myErr != noErr)
        goto bail;

    // elicit an SDP file from the user
    myFileFilterUPP = QTFrame_GetFileFilterUPP
        ((ProcPtr)QTFrame_FilterFiles);
    myErr = QTFrame_GetOneFileWithPreview(myNumTypes,
        (QTFrameTypeListPtr)myTypeList, &myFSSpec,
        myFileFilterUPP);
    if (myErr != noErr)
        goto bail;

    // start broadcasting from an SDP file
    myPresParamsPtr->dataType = kQTSFileTypeData;
    myPresParamsPtr->data = &myFSSpec;
    myPresParamsPtr->dataLength = sizeof(myFSSpec);

    // set the presentation flags: use Sequence Grabber, don't display blue Q movie,
    // and send data
    myPresParamsPtr->flags = kQTSAutoModeFlag |
        kQTSdontShowStatusFlag | kQTSsendMediaFlag;

    myPresParamsPtr->timeScale = 0;
    myPresParamsPtr->mediaParams = myMediaParamsPtr;
}
```

```
// fill these in to get status notifications
myPresParamsPtr->notificationProc = gNotificationUPP;
myPresParamsPtr->notificationRefCon = 0L;

// define the display size and the default transmission size
myPresParamsPtr->mediaParams->v.width = Long2Fix(176);
myPresParamsPtr->mediaParams->v.height = Long2Fix(144);

TranslateMatrix
    (&(myPresParamsPtr->mediaParams->v.matrix),
    Long2Fix(10), Long2Fix(10));

// set the window that Sequence Grabber will draw into
myPresParamsPtr->mediaParams->v.gWorld =
    GetDialogPort(gMonitor);
myPresParamsPtr->mediaParams->v.gdHandle = NULL;

// create a new presentation
myErr = QTSNewPresentation(myPresParamsPtr,
    &gPresentation);
if (myErr != noErr)
    goto bail;

myErr = QTSPresPreview(gPresentation, kQTSAAllStreams,
    NULL, kQTSNormalForwardRate, 0);

bail:
    if (myPresParamsPtr != NULL)
        DisposePtr((Ptr)myPresParamsPtr);
    if (myMediaParamsPtr != NULL)
        DisposePtr((Ptr)myMediaParamsPtr);
    if (myFileFilterUPP != NULL)
        DisposeNavObjectFilterUPP(myFileFilterUPP);

    return(myErr);
}
```

Notice that we dispose of the new presentation parameters structure (`myPresParamsPtr`) and the media parameters structure (`myMediaParamsPtr`) before returning from `QTBC_SetupPresentation`. `QTSNewPresentation` takes care to copy all the information we pass it, including any blocks of memory referenced in the new presentation parameters structure. If we had allocated any other information (for instance, a clip region), we should also dispose of that memory too.

BROADCASTING

Now we've created a presentation and requested that the video data that is to be broadcast be previewed in the monitor window. In order for the previewing and subsequent broadcasting to proceed, we need to grant QuickTime some time to process the presentation. We do this by calling the `QTSPresIdle` function periodically. As usual, we'll add some code to our application's `QTApp_Idle` function, which is called whenever we receive a null event:

```
if (gPresentation != kQTSInvalidPresentation)
    QTSPresIdle(gPresentation, NULL);
```

This just says: if we have an active presentation, grant it some processor time.

Starting the Broadcast

So far, however, no broadcasting is happening. In `QTBroadcast`, we wait until the user clicks the Start button in the monitor window to begin that process. When the user clicks that

Now Up-to-Date & Contact®

The #1 Best-Selling Group Scheduler and Contact Manager for the Macintosh!

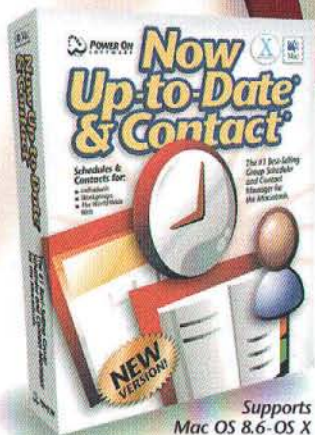
You're a busy person. You've got things to do, people to see, and a seemingly infinite list of things to keep track of. Whether it's just yourself or a department at a Fortune® 500 company, you (and everyone that works with you) need to be organized. Maybe you've tried other ways: paper planners, lots of sticky notes, other software. But now you can't afford to waste any more time or effort. You need a serious solution. One that's so easy to use that you can get started right out of the box. One that will grow with you. One that has the features and capabilities you need.

Now Up-to-Date & Contact is the internet-savvy contact manager. With a click of the mouse, you will be soaring through cyberspace. Want to publish your calendar on the web, or get a map to a meeting, or even find the restaurant nearest to an important client? It's all a snap with Now Up-to-Date & Contact! Additional features

you'll rave about include: built-in word processing with mail merge, auto-mated fax creation, integration with your favorite email software, advanced Palm synchronization, and the amazing new Grab-'n-Go™ that allows you to instantly grab information from virtually any source and have Now Up-to-Date & Contact create any necessary follow-up reminders. Now Up-to-Date & Contact even understands English commands so you can schedule appointments without touching the keyboard.*

With its amazing power and phenomenal speed, combined with unparalleled ease of use, it is no wonder that Now Up-to-Date & Contact has won the coveted Eddy Award for best Information Manager, seven World Class Awards, and a host of rave reviews. Now, with Version 4.1, Now Up-to-Date & Contact adds full support for Mac OS X, making it the indispensable product for people who want to be organized.

**Made for
Mac OS X**



Supports
Mac OS 8.6-OS X



Phone: 800-344-9160 • 614-413-4000 Fax: 614-413-4100 • www.poweronsoftware.com

Copyright © 2001 by Power On Software, Inc. All rights reserved. Now Up-to-Date & Contact is a registered trademark of Power On Software, Inc. Macintosh is a registered trademark of Apple Computer, Inc. Palm is a registered trademark of Palm Computing. Your is a registered trademark of Pandapring. All other trademarks acknowledged.



button, QTBroadcast calls the QTBC_StartBroadcasting function, defined in Listing 4. (Later we'll take a look at the code that handles those button clicks.)

Listing 4: Starting broadcasting

```
QTBC_StartBroadcasting
OSErr QTBC_StartBroadcasting (void)
{
    // stop the preview
    QTSPresPreview(gPresentation, kQTSAllStreams, NULL,
        kQTSStoppedRate, 0);

    return(QTSPresPreroll(gPresentation, kQTSAllStreams, 0,
        (Fixed)kQTSNormalForwardRate, 0L));
}
```

QTBC_StartBroadcasting first calls QTSPresPreview with a rate of kQTSStoppedRate to stop previewing the data; then it calls QTSPresPreroll, which readies the media for subsequent broadcasting and performs any necessary handshaking between the transmitter (that is, the broadcasting computer) and the client computer or computers. (As you've probably guessed, prerolling a presentation with QTSPresPreroll is analogous to prerolling a movie with PrerollMovie.)

QTSPresPreroll returns fairly quickly, as some of its operations involve establishing network connections and hence should occur asynchronously. We get informed of the success of the prerolling in our presentation notification procedure, by receiving a notification message of type kQTSPrerollAckNotification. Once we receive this acknowledgement, we can then begin actually broadcasting data by calling the QTSPresStart function. Listing 5 shows our complete presentation notification procedure.

Listing 5: Handling notification messages

```
QTBC_NotificationProc
static PASCAL_RTN ComponentResult QTBC_NotificationProc
(ComponentResult theErr, OSType theNotificationType,
    void *theNotificationParams, void *theRefCon)
{
    #pragma unused(theErr, theNotificationParams, theRefCon)
    QTSPresentation myPresentation = kQTSInvalidPresentation;
    ComponentResult myErr = noErr;

    switch (theNotificationType) {
        case kQTSNewPresentationNotification:
            // when we get this notification, the presentation has been created
            // and is sent to us in theNotificationParams;
            // if we needed it, we could retrieve it as follows:
            myPresentation =
                (QTSPresentation)theNotificationParams;
            break;

        case kQTSPrerollAckNotification:
            myErr = QTSPresStart(gPresentation, kQTSAllStreams,
                0L);
            break;

        case kQTSStartAckNotification:
        case kQTSStopAckNotification:
            break;

        default:
            break;
    }

    return(myErr);
}
```

As you can see, we ignore most of the notification messages. We handle the kQTSPrerollAckNotification message, to start broadcasting once the prerolling is complete. We also handle the kQTSNewPresentationNotification message, which is sent whenever a new presentation is created; in this case, the identifier for the new presentation is passed in the theNotificationParams parameter. (We don't actually do anything with that identifier; this code is provided to illustrate how to retrieve it if you need it.)

Finally we're off and running. We've created a presentation, prerolled it, and (hopefully) received a notification that the prerolling was successful — in which case we've called QTSPresStart to start broadcasting data. Now it's up to the client to start receiving the data we're transmitting. This can happen in one of several ways. The easiest way is to provide the client with a copy of the SDP file that we used to initiate the broadcast session. If the user opens this file with a QuickTime-savvy application, then QuickTime's *SDP importer* is called to convert the SDP data into a movie. When the application opens that movie, QuickTime connects to the server, requests that it start sending it data, and then receives the data into a local buffer; while this buffering is happening, the user will see a connection status display like the one shown in Figure 7.

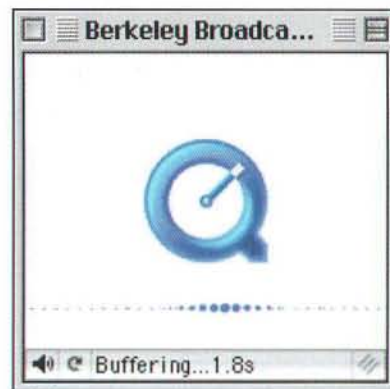


Figure 7: A client status message

The user can then save the movie into a new file; in this case, the new file is a QuickTime movie file that contains a streaming track; the media data for this track is just the text data in the original SDP file. Subsequently opening the movie file is pretty much identical to opening the SDP file.

Controlling the Broadcast

So we now see how to start broadcasting from a transmitter and how to receive the broadcast stream on a client. If we want to pause the broadcasting, we can call the QTBC_PauseBroadcasting function defined in Listing 6. This function calls QTSPresStop to stop the presentation and then QTSPresPreview with a rate of kQTSNormalForwardRate to resume previewing in the monitor window.

Sometimes, one just isn't enough...

DVIator for ADC



... and sometimes it is.

With DVIator and a DVI video card you can add additional ADC displays to a new G4, or upgrade an older Mac with Apple's new flat-panels.



Dr. Bott OFF

www.drbott.com • 877.611.2688 • 503.582.9944

Listing 6: Pausing broadcasting

```
QTBC_PauseBroadcasting
OSErr QTBC_PauseBroadcasting (void)
{
    OSErr    myErr = noErr;

    myErr = QTSPresStop(gPresentation, kQTSA11Streams, 0L);
    if (myErr != noErr)
        goto bail;

    // restart the preview
    myErr = QTSPresPreview(gPresentation, kQTSA11Streams, NULL,
        kQTSNormalForwardRate, 0);

bail:
    return(myErr);
}
```

After pausing a broadcast by calling `QTBC_PauseBroadcasting`, we could later resume the broadcast by calling `QTBC_StartBroadcasting` once again. If we want to stop broadcasting for good (perhaps because the application is shutting down), we can call the `QTBC_StopBroadcasting` function defined in Listing 7. This function calls `QTSPresStop` to stop broadcasting and then `QTSDisposePresentation` to dispose of the presentation. Notice that we also set the global variable `gPresentation` to `kQTSInvalidPresentation`, so that `QTSPresIdle` is no longer called in `QTApp_Idle`.

Listing 7: Stopping broadcasting

```
QTBC_StopBroadcasting
OSErr QTBC_StopBroadcasting (void)
{
    OSErr    myErr = noErr;

    if (gPresentation != kQTSInvalidPresentation) {
        myErr = QTSPresStop(gPresentation, kQTSA11Streams, 0L);

        if (myErr != noErr)
            myErr = QTSDisposePresentation(gPresentation, 0L);

        gPresentation = kQTSInvalidPresentation;
    }

    return(myErr);
}
```

BROADCAST SETTINGS

Once we've created a new presentation based on an existing SDP file, we want to allow the user to view and configure the presentation's settings. `QTBroadcast` provides the "Configure Settings..." menu item in its Test menu; when the user selects this item and a presentation is underway, `QTBroadcast` displays the Transmission Settings dialog box shown in **Figure 8**.

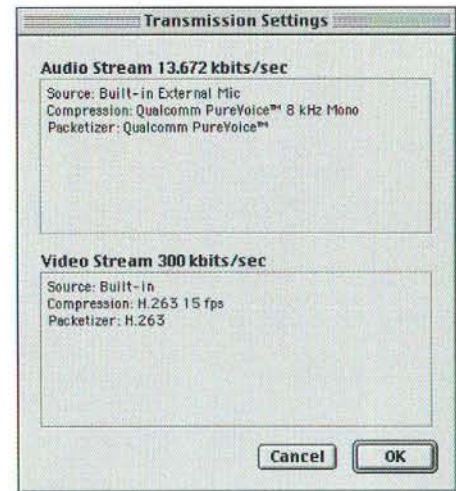


Figure 8: The Transmission Settings dialog box

This dialog box shows the source, compression algorithm, and packetizer currently being used for the audio and video streams, and also displays the current data rate for each stream. Notice that there is no way for the user to modify any of these settings, since the broadcast has already begun.

If a presentation is not underway, then when the user selects the "Configure Settings..." menu item, `QTBroadcast` displays the Transmission Settings dialog box shown in **Figure 9**.

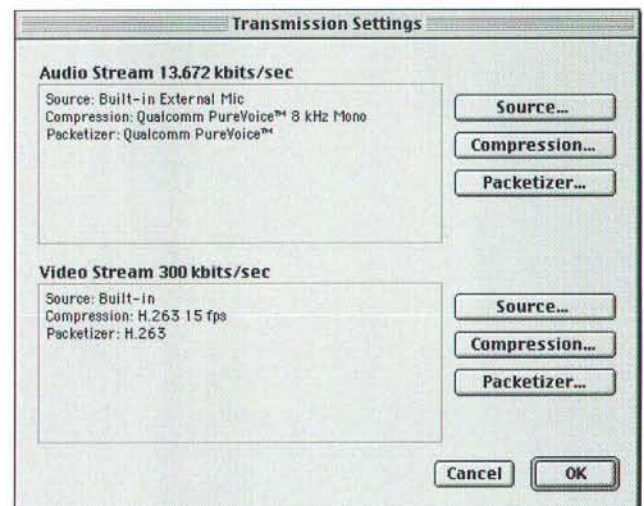


Figure 9: The Transmission Settings dialog box (modifiable)

As you can see, the dialog box now contains some buttons that allow the user to modify various broadcast settings. For instance, if the user clicks the "Source..." button associated with the video stream, the dialog box shown in **Figure 10** appears.

Without this piece, you're not done.



It's not enough just to write solid code anymore. Users expect an easy-to-use installer when they buy your software.

InstallerMaker gives you all the tools you need to install, uninstall, resource-compress or update your software in one complete, easy-to-use package.

Add marketing muscle to your installers by customizing your electronic registration form to include surveys and special offers. Even create demoware in minutes.

Visit <http://www.stuffit.com/installermaker/> to also learn about creating Mac® OS X compatible installers with InstallerMaker.

Then you're done.

Stuffit InstallerMaker

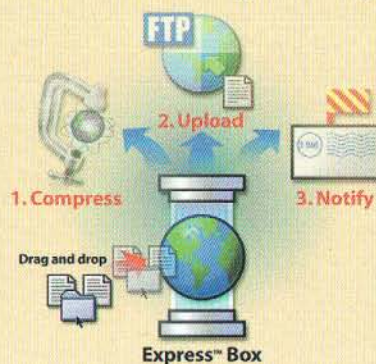
The complete installation and distribution solution.™

Get your beta testing started faster.

Guarantee your builds arrive safe and sound! New **Stuffit Express™** automates file compression and transfer tasks to eliminate costly errors every time you exchange data. By creating custom drop box applications, you and those you communicate with can accomplish complex file transfer tasks with a simple drag and drop. Choose from up to 26 file transfer steps with no coding required. Boost

productivity and save time by distributing your custom Express boxes to everyone on your list.

<http://www.stuffit.com/express/enterprisemac>.



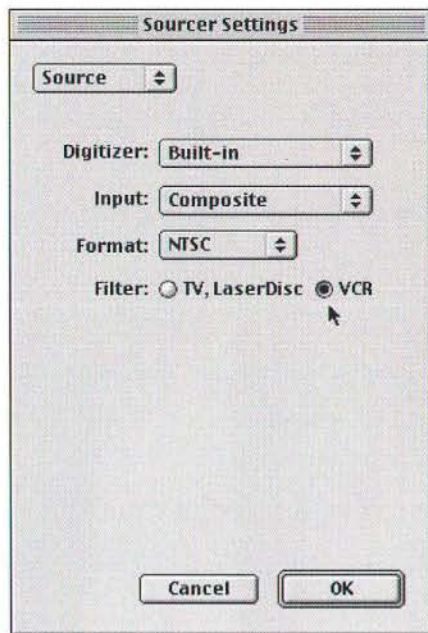


Figure 10: The video stream Sourcer Settings dialog box

And if the user clicks the “Packetizer...” button associated with the video stream, the dialog box shown in Figure 11 appears.

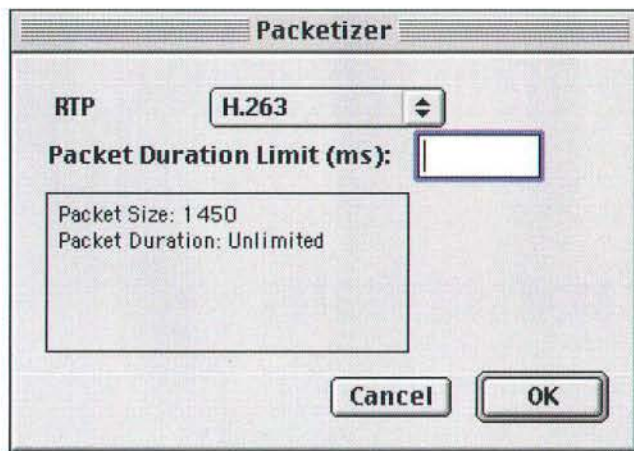


Figure 11: The video stream Packetizer dialog box

It's actually quite simple to display all these dialog boxes; indeed, we need to call only a single function, `QTSPresSettingsDialog`, as shown in Listing 8 (which lists the code we execute in response to the “Configure Settings...” menu item).

Listing 8: Handling the “Configure Settings...” menu item

```
QTApp_HandleMenu
case IDM_GET_SETTINGS:
    if (gPresentation != kQTSInvalidPresentation)
        myErr = QTSPresSettingsDialog(gPresentation,
            kQTSAAllStreams, 0, NULL, 0L);

    if (myErr != noErr)
        QTFrame_Beep();
    myIsHandled = true;
    break;
```

MONITOR WINDOW CONTROL

We're just about finished learning how to use QuickTime to broadcast data across a network. We've worked with all the basic functions we need to use to create and preroll presentations, start and stop previewing, and actually start sending streams of data to a remote client. Let's finish off quickly by taking a look at the code we use to create and manage the monitor window.

We create a single monitor window at application launch time by calling the `QTBC_CreateMonitorWindow` function, defined in Listing 9. This function calls `GetNewDialog` to open a dialog box whose attributes and items are defined by a 'DLOG' resource of ID `kMonitorDLOGID`.

Listing 9: Creating the monitor window

```
QTBC_CreateMonitorWindow
DialogPtr QTBC_CreateMonitorWindow (void)
{
    DialogPtr myDialog = NULL;

    myDialog = GetNewDialog(kMonitorDLOGID, NULL,
        (WindowPtr)-1L);

    if (myDialog != NULL) {
        short myItemKind;
        Handle myItemHandle = NULL;
        Rect myItemRect;

        MacSetPort(GetDialogPort(myDialog));

        // set the user item drawing procedure
        GetDialogItem(myDialog, kMonitorUserItemID, &myItemKind,
            &myItemHandle, &myItemRect);
        SetDialogItem(myDialog, kMonitorUserItemID, myItemKind,
            (Handle)gMonitorUserItemProcUPP, &myItemRect);

        MacShowWindow(GetDialogWindow(myDialog));
        QTBC_UserItemProcedure(myDialog, kMonitorUserItemID);
    }

    return(myDialog);
}
```

`QTBC_CreateMonitorWindow` also sets the user item procedure for the user item in the monitor window to be the function `QTBC_UserItemProcedure`, defined in Listing 10. This user item procedure draws a frame around the user item and also sets the state of the Start/Pause button; if no presentation is active, we want to disable the button so that it cannot be pressed.

Listing 10: Drawing the user item in the monitor window

```
QTBC_UserItemProcedure
PASCAL_RTN void QTBC_UserItemProcedure (
    DialogPtr theDialog, short theItem)
{
    short myItemKind;
    Handle myItemHandle = NULL;
    Rect myItemRect;
    GrafPtr mySavedPort;
    OSErr myErr = noErr;

    GetPort(&mySavedPort);
    MacSetPort(GetDialogPort(theDialog));

    if (theItem != kMonitorUserItemID)
        goto bail;

    // draw a frame around the user item rectangle
    GetDialogItem(theDialog, kMonitorUserItemID, &myItemKind,
        &myItemHandle, &myItemRect);
    InsetRect(&myItemRect, -1, -1);
    MacFrameRect(&myItemRect);
}
```



```
// enable the Start/Stop button in the monitor window
GetDialogItem(theDialog, kMonitorButtonID, &myItemKind,
    &myItemHandle, &myItemRect);
if (myItemHandle != NULL) {
    if (gPresentation == kQTSInvalidPresentation)
        HiliteControl((ControlHandle)myItemHandle, 255);
    else
        HiliteControl((ControlHandle)myItemHandle, 0);
}

bail:
    MacSetPort(mySavedPort);
}
```

We handle clicks on the button in the QTBC_HandleMonitorWindowEvents function (Listing 11), which is called by QTApp_HandleEvent for any events targeted at a dialog box.

Listing 11: Handling events in the monitor window

```
QTBC_HandleMonitorWindowEvents
(DialogPtr theDialog, DialogItemIndex theItemHit)
{
    short    myItemKind;
    Handle    myItemHandle = NULL;
    Rect      myItemRect;

    if ((theDialog == gMonitor) &&
        (theItemHit == kMonitorButtonID)) {

        GetDialogItem(theDialog, kMonitorButtonID, &myItemKind,
            &myItemHandle, &myItemRect);
        if (gBroadcasting) {
            QTBC_PauseBroadcasting();
            SetControlTitle((ControlHandle)myItemHandle,
                "\pStart");
        } else {
            QTBC_StartBroadcasting();
            SetControlTitle((ControlHandle)myItemHandle,
                "\pPause");
        }

        gBroadcasting = !gBroadcasting;
    }
}
```

CONCLUSION

The broadcasting support included in QuickTime 5 allows us to write applications that transmit data captured or stored on a local server to a client (or set of clients) located remotely on a network. It's important to remember that the streams of media data transmitted by QuickTime's broadcasting components conform to the IETF protocols (principally, RTP and RTSP) and hence can be received by any standards-compliant client application. As we've seen, QuickTime 4 provided support for receiving those kinds of streams, so we can use QuickTime APIs to develop both the transmitter and the receiver.

CREDITS

Thanks are due to **Kevin Marks** for reviewing this article and providing some comments, and to **Anne Jones** for providing some helpful background information.

Tri-platform Calendaring

OS X *Carbon*

Mac OS 7.1-9.x

Windows 95-2000

New!

CalendarMonster 1.2

Multi-platform calendar



• End-user Calendar

The best FMP calendar experience

- Fast, fast, fast!!!
- Platform-appropriate user experience
- Highly scalable

• Excellent Developer Support

Quickly and easily integrate with your FileMaker Pro projects

- 100% live data, 100% modular, no plug-ins
- FileMaker Pro 4 or 5 (including 5.5)
- As little as 2 ScriptMaker calls
- Dedicated developers' utility



Developer and volume licensing available



ASCENDING
TECHNOLOGIES

monster@asctech.com

Download your own FREE fully-functional demo

<http://www.asctech.com>

by Bob Boonstra, Westford, MA

MEGAMINX

Some months the Challenge ideas come easily. Other months an interesting idea eludes me. This is one of those "other" months. The column deadline looms near, becomes imminent, encroaches into the present, overtakes me, and rushes onward, leaving me in its past. The wrath of my editor awaits. What to do?

Just in time, my collection of mechanical puzzles beckons, and an idea pops into my head. Time for another Meffert puzzle, perhaps. Something more difficult than the Tetraminx Challenge from June of 1999. Something even more difficult than the R*biks Cube Challenge. Yes, this month we'll Challenge readers to solve the Megaminx, pictured below.



The Megaminx is a regular solid with twelve faces, where each face is divided into one center face element (which I'll call a facelet), five corner facelets, and five edge facelets. The faces are colored with six colors, pairs of opposing faces being colored the same. The solid is divided into slices corresponding to the faces of the Megaminx. Each face can be rotated clockwise or counterclockwise, with five rotation steps in the same direction bringing the corresponding slice back to its original position. Your Challenge is to restore a scrambled Megaminx to its original position. Entries this month will be complete applications, so there is no prototype for the code you should write.

To describe the coloring of the solid, we're going to have to agree on some notation to identify the facelets. We'll number the faces from 1..12, thinking of face 1 as the "top" face. Face 2 will be one of the five faces adjacent to the top face, and faces 3..6 the other faces adjacent to the top face, proceeding clockwise from face 2 when viewed from above the top face. Faces 7..12 will be numbered such that face N+6 is opposite face N. We'll number the colors of the solved Megaminx using 1..6 as well, assigning color N to the center facelet of faces N and N+6. Sounds complicated perhaps, but if you read carefully, it should become clear. If it isn't, perhaps the following list of face adjacencies will help:

Face 1 is adjacent to faces:	2	3	4	5	6
Face 2 is adjacent to faces:	6	10	11	3	1
Face 3 is adjacent to faces:	2	11	12	4	1
Face 4 is adjacent to faces:	3	12	8	5	1
Face 5 is adjacent to faces:	4	8	9	6	1
Face 6 is adjacent to faces:	5	9	10	2	1
Face 7 is adjacent to faces:	8	9	10	11	12
Face 8 is adjacent to faces :	12	4	5	9	7
Face 9 is adjacent to faces:	8	5	6	10	7
Face 10 is adjacent to faces:	9	6	2	11	7
Face 11 is adjacent to faces:	10	2	3	12	7
Face 12 is adjacent to faces:	11	3	4	8	7

Your program must process multiple test cases, the number of which is provided in the file `input.txt`. The initial conditions for each test case are provided in file `megaminxNN.in`, where NN ranges from 1 to the number of test cases. Each input file contains 120 lines, 60 describing the coloring of the corner facelets, followed by 60 describing the coloring of the edge facelets. The lines describing the corner facelets will contain four comma-delimited numbers: the number of the face on which the facelet lies, the number of one of the other faces on this corner piece, the number of the third face on this corner piece, and the color of the facelet. The lines describing the edge facelets will contain three comma-delimited numbers: the number of the face on which the facelet lies, the number of the other face defining this corner piece, and the color of the facelet. So, for a Megaminx in the "solved" orientation, the `megaminxNN.in` file would contain the following lines for the top face:

```
1,2,3,1
1,3,4,1
1,4,5,1
1,5,6,1
1,6,2,1
...
1,2,1
1,3,1
1,4,1
1,5,1
1,6,1
...
```

The puzzle is solved with a sequence of slice rotations, resulting in each facelet being moved to the face that matches its color. Each rotation moves the corner and edge pieces of the rotated slice to new positions. A clockwise rotation of the slice corresponding to face 1, for example, moves corner piece (1,2,3) to (1,3,4), which moves to (1,4,5), which moves to (1,5,6), which moves to (1,6,2). A clockwise rotation of slice 2 moves the corners from (2,3,1) -> (2,1,6) -> (2,6,10) -> (2,10,11) -> (2,11,3) -> (2,3,1).

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."

— MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."

— MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."

— Leonard Rosenthal, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"

— Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."

— Bill Goodman, author of *Smaller Installer* and *Compact Pro*

"Resorcerer Rocks! Buy it, you will NOT regret it."

— Joe Zobkiw, author of *A Fragment of Your Imagination*

"Resorcerer will pay for itself many times over in saved time and effort."

— MacUser review

"The template that disassembles 'PICT's is awesome!"

— Bill Steinberg, author of *Pyro!* and *PBTools*

"Resorcerer proved indispensable in its own creation!"

— Doug McKenna, author of *Resorcerer*



Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: \$256 (decimal)

Website price: \$128 - \$256

(Educational, quantity, or
other discounts available)

Includes: Electronic documentation

60-day Money-Back Guarantee

Domestic standard shipping

Payment: Check, PO's, or Visa/MC

Taxes: Colorado customers only

Extras (call, fax, or email us):

COD, FedEx, UPS Blue/Red,

International Shipping

MATHEMAESTHETICS, INC.

PO Box 298

Boulder, CO 80306-0298 USA

Phone: (303) 440-0707

Fax: (303) 440-0504

resorcerer@mathemaesthetics.com

**New
in
2.0:**

- Very fast, HFS browser for viewing file tree of all volumes
- Extensibility for new Resorcerer Apprentices (CFM plug-ins)
- New AppleScript Dictionary ('aete') Apprentice Editor
- MacOS 8 Appearance Manager-savvy Control Editor
- PowerPlant text traits and menu command support
- Complete AIFF sound file disassembly template
- Big-, little-, and even mixed-endian template parsing
- Auto-backup during file saves; folder attribute editing
- Ships with PowerPC native, fat, and 68K versions

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'icb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

After reading the input for each test case, you should determine the sequence of rotations needed to solve the puzzle and output them to a file `rotationsNN.txt`. Each rotation should produce one line of output of the form:

```
face,direction
```

... where `face` is the number of the slice being rotated, and `"direction"` is the character `+` for a clockwise rotation and `-` for a counterclockwise rotation.

Finally, your program should produce a `megaminx.log` file, with one line per test case containing the integer number of microseconds used by your application to solve that test case, including the time to read the input, find the solution, and produce the `rotationsNN.txt` output.

Scoring will be based on minimizing execution time. You can earn a reduction of up to 25% of your execution time if your application has an option to display the Megaminx as it is being solved. (Scoring will be done with the display option disabled, so the display time will not count against your solution.) The bonus will be awarded based on a subjective evaluation of the quality and attractiveness of your application.

This will be a native PowerPC Challenge, using the development environment of your choice, provided I have or can obtain a copy - email progchallenge@mactech.com to check before you start. You can develop for Mac OS 9 or Mac OS X. Your solution should be a complete Macintosh application, and your submission should provide everything needed to build your application.

WINNER OF THE DECEMBER, 2001 CHALLENGE

Congratulations to **Ernst Munter** (Kanata, ON, Canada) for winning the December Parent-Teacher Conferences Challenge. This problem required contestants to schedule appointments between parents of children with their teachers. Penalty points were assessed for each requested conference that was not scheduled, for any scheduling gaps that caused parents or teachers to be idle between scheduled conferences, as well as for elapsed execution time. Parents indicate the strength of their desire for a conference with a teacher of a given child, and the penalty for not scheduling a conference is proportional to the parents' desire to hold one.

Ernst starts with an initial assignment of conferences that gives priority first to those parents whose availability is low, and then to those conferences that are most strongly desired by parents. He then tries to reduce scheduling gaps by moving conferences to different slots where both parent and teacher are available, assigning the conference to the other parent of the child, or swapping time slots with another parent who has a conference with the same teacher. The

solution iterates on these techniques until no further advantage is obtained, after which it attempts to reduce penalties for scheduling gaps by canceling conferences entirely. Ernst not only achieved the lowest number of penalty points, but did so using the least execution time.

Alan Stenger again shares the prize with Ernst as the best-placing entrant without a recent victory. Alan's entry incurred about 9% more penalty points, in part because of two factors. First, he did not recognize cases when one parent had requested a conference for multiple children with the same teacher, correctly observing that this was unlikely to happen often in the real world. Second, he did nothing special to reduce penalties due to scheduling gaps, beyond scheduling conferences starting from the middle of the available periods and working outward from there.

Tom Saxton took second place (but as a recent winner does not share in the prize). Tom also processes conference requests in order of the strength of the parents' desire to hold it, moving other conferences held by a teacher if necessary and possible, or rescheduling the parent's conferences. Tom's approach incurred about 6% more penalty points than Ernst's.

Acknowledgements also to Ken Slezak and Tetuya Takeo for submitting solutions in non-Metrowerks development environments, Ken for using REALBasic, and Tetuya for using Objective C.

The table below lists, for each of the solutions submitted, the number of penalty points assessed due to missed conferences and scheduling gaps, the total execution time in milliseconds, and the penalty points after adding 10% for each second of execution time per test case. It also lists the programming language of each entry. As usual, the number in parentheses after the entrant's name is the total number of Challenge points earned in all Challenges prior to this one.

Name	Penalty	Time (msec)	Points	Language
Ernst Munter (800)	48513	571.4	48825.0	C++
Tom Saxton (193)	51476	760.0	51997.1	C++ (OS X)
Alan Stenger (75)	52744	1482.8	53636.5	C++
Tetuya Takeo	54224	8539.3	60256.7	Objective C
Jan Schotsman (14)	59745	2192.0	61364.1	C
Ken Slezak (26)	60046	8199.0	66946.7	REALBasic

TOP CONTESTANTS ...

Listed here are the Top Contestants for the Programmer's Challenge, including everyone who has accumulated 20 or more points during the past two years. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

A New System Has Arrived.

**Don't Get Caught
With Your
Mac Down!**



No Mac is complete without Timbuktu Pro, the premier remote control and file transfer software for Mac OS for over ten years.

No network is complete without the smart systems management of netOctopus.

Both tools are newly rewritten for OS X, bringing the power and simplicity of Netopia software to the world's most advanced operating system. For deployment, training and support of all your Macs (and PC's!) Timbuktu and netOctopus are indispensable.

Are you ready to migrate? We're ready to take you there.

macosxready.com

Timbuktu Pro

netopia

netOctopus

Rank	Name	Points (24 mo)	Wins (24 mo)	Total Points
1.	Munter, Ernst	265	10	822
2.	Rieken, Willeke	73	3	134
3.	Wihlborg, Claes	49	2	49
4.	Saxton, Tom	45	1	203
5.	Taylor, Jonathan	39	1	63
7.	Maurer, Sebastian	31	1	108
9.	Mallett, Jeff	20	1	114
10.	Cooper, Tony	20	1	20
11.	Truskier, Peter	20	1	20

... AND THE TOP CONTESTANTS LOOKING FOR A RECENT WIN

In order to give some recognition to other participants in the Challenge, we also list the high scores for contestants who have accumulated points without taking first place in a Challenge during the past two years. Listed here are all of those contestants who have accumulated 6 or more points during the past two years.

Rank	Name	Points (24 mo)	Total Points
6.	Boring, Randy	32	144
8.	Sadetsky, Gregory	22	24
12.	Shearer, Rob	19	62
13.	Stenger, Allen	17	82
14.	Schotsman, Jan	16	16
15.	Nepsund, Ronald	10	57
16.	Hart, Alan	10	35
17.	Day, Mark	10	30
18.	Downs, Andrew	10	12
19.	Desch, Noah	10	10
20.	Fazekas, Miklos	10	10
21.	Flowers, Sue	10	10
22.	Leshner, Will	7	7
23.	Miller, Mike	7	7

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points
2nd place	10 points
3rd place	7 points
4th place	4 points
5th place	2 points
finding bug	2 points
suggesting Challenge	2 points

Here is Ernst's winning Parent-Teacher Conferences solution:

PT-Main.cpp

Copyright © 2001

Ernst Munter, Kanata ON, Canada

/*
The task

Schedule parent-teacher conferences, while maximizing parent and teacher satisfaction. Each conference relating to a child involves a teacher, a parent, and a priority. Satisfaction is maximized by accommodating those conferences having higher priority while minimizing the time any teacher or parent have to wait between conferences. The input is provided in the form of three text files for each test case. The output is a text file listing the conferences, and a log file reporting the run times.

Solution

Each input file is scanned to generate an array of corresponding records of children, parents, and teachers. Names are replaced by pointers linking parents to children, and parents and teachers to conference requests. Spouses are linked to each other.

Conference requests for children with the same parents and teacher reference, are combined, their priorities added together and transferred to a single request. All redundant requests are then removed.

Conference requests are first sorted to place the highest priority requests at the front. The first N requests are then sorted to favour requests for conferences with parents who have less time available. N is the number of teachers multiplied by the number of available conference periods.

A processing sequence of the conference periods is prepared, based on the total availability of parents for each period. Periods with less availability will be used first in trying to accommodate a request.

Then all requests are sequentially served, and if possible accommodated by booking a conference period with the requested teacher.

When no more matches can be found, a loop of simple strategies is used to reduce the waiting penalties, essentially by shifting conferences around among time slots, parents and their spouses, and between parents having conferences with the same teacher.

When no more improvement can be made, the set of conferences is sent to the disk file.

Version 2 addition: a 4th wait-reduction technique is added by which conferences may be cancelled if the waiting times eliminated exceed the priority points lost.

Version 3 addition: Exploits the rule that a spouse may use a parent in the same conference, to reduce a spouse's waiting time. In the test cases, this change resulted in only a tiny improvement. The data structure does not allow the recognition of more than one spouse, so there may still be some missed opportunities for points savings.

Limitations

Where there are two parents to a child, they are usually "married" and cross linked in the database. However, no parent can have more than one spouse, and only one crosslinkage can exist. This implies that we can use spousal cross links to find alternates for conferences, but we cannot infer from the absence of a crosslink that (another) spouse does not exist.

No multiple conferences can occur with the same teacher at the same time, i.e. only one parent can be present at a conference. This restriction was removed in version 3.

Assumptions

The number of periods does not exceed 255. This limit can be increased in the PT-compile-constants header file.

(It might have been nice to use 32-bit bit maps in some cases, but I thought 32 might be a bit too low as a realistic limit for the general case)

Associated Program Files

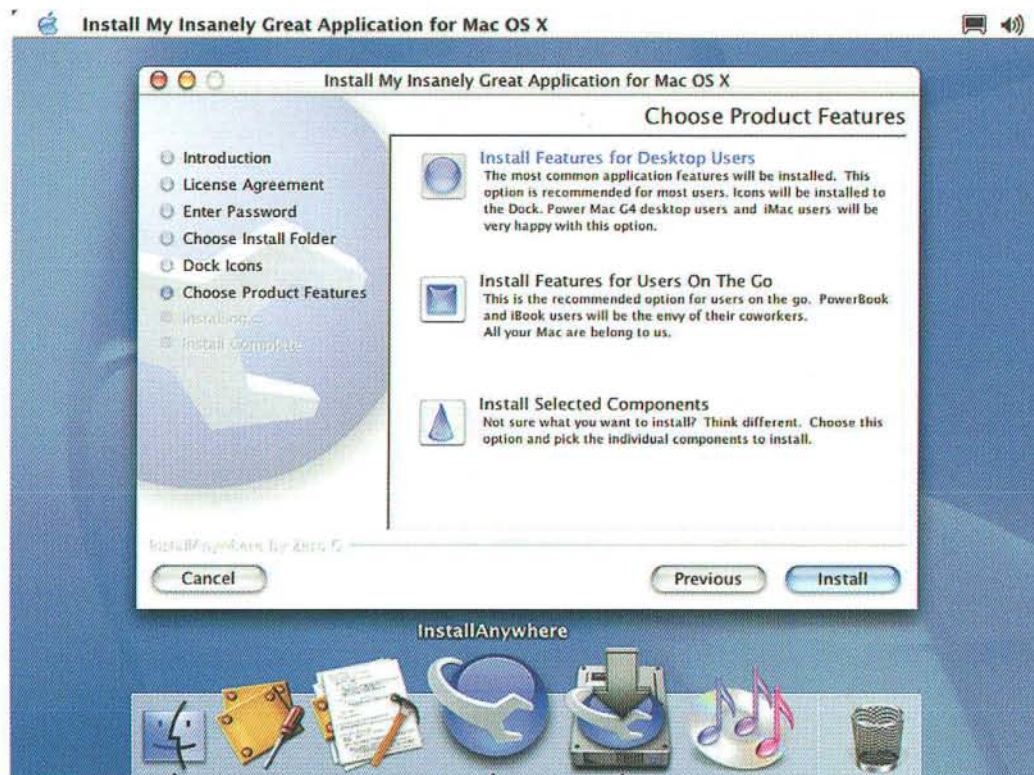
The main program (this file) is "PT-Main.cp".

The header file "PT-Files.h" contains classes for the various input and output files.

The header file "PT-Persons.h" contains classes for the person records.

The header file "PT-compile-constants.h" contains a few compile policies, such as the maximum number of periods (defaulting to 255), and the debug and assert controls. It also contains a directory string which can be used to locate the input and

LOOKING GOOD



Looking good on Mac OS X is easy with InstallAnywhere!

Let's face it, first impressions are lasting ones. You don't build beautiful applications for Mac OS X only to deliver them with yesterday's tired installers.

Zero G has created a powerful and intuitive installation and configuration solution that is true to today's Aqua look and feel.

InstallAnywhere has complete support for Mac OS X, such as authentication, file permissions, and platform specific locations including installing icons into the Dock, making it the only installer you'll ever need.

And when you're ready to deploy to Mac OS Classic, Linux, Solaris, AIX, HP-UX, or even <gasp!> Windows, InstallAnywhere is ready too - anytime, anywhere, on any platform.

Software innovators like Adobe, Apple, BEA, Borland, HP, i2, Iona, Sun, TIBCO and Xerox already depend on InstallAnywhere for their software deployment needs.

See for yourself why InstallAnywhere, the world's leading multi-platform deployment solution, is also the best looking and most powerful installer for Mac OS X.



Download a free trial version from <http://www.ZeroG.com/goto/macosx>



© 2001 Zero G Software, Inc. InstallAnywhere and Zero G are trademarks or registered trademarks of Zero G Software, Inc. Mac is a trademark of Apple Computer, Inc., registered in the U.S. and other countries. The "Built for Mac OS X" graphic is a trademark of Apple Computer Inc., used under license. All other trademarks are property of their respective owners.

output files in a directory other than the executable.

The header file "PTUtilities.h" and the file "PTUtilities.cpp" contain some minor utility functions, not all of which may actually be still in use in this version.

The header file "MyTimer.h" contains a microsecond timer class.

```

*/
#include "PT-Persons.h"
#include "PT-Files.h"
#include "PT-compile-constants.h"
#include "PT-Utilities.h"
#include "MyTimer.h"

class AvailabilityByPeriod
// Used in computing the optimal period assignment order.
{
    long availabilitySum; // of all parents
    long period;
public:
    void Init(long x) {availabilitySum=0;period=x;}
    void Increment() {availabilitySum++;}
    long Period() const {return period;}
    int operator - (AvailabilityByPeriod& r) const
    {return availabilitySum-r.availabilitySum;}
};

class TestCase
// Contains the resources of a test case, files, parents, teachers, etc.
{
    SchedulesFile schedulesFile;
    ChildrenFile childrenFile;
    TeachersFile teachersFile;
    OutputFile outputFile;

    long numPeriods;

    Parent* parents;
    long numParents;

    Child* children;
    long numChildren;

    Request* requests;
    long numRequests;

    Teacher* teachers;
    long numTeachers;

    Conference* conferences;
    long numConferences;

    Period_t* assignmentOrder;
public:
    TestCase(long testNumber) :
// Constructor reads all input files and sets up the resources.
        schedulesFile(testNumber,numPeriods,parents,numParents),
        childrenFile(testNumber,children,numChildren),
        teachersFile(testNumber,requests,numRequests),
        outputFile(testNumber),
        teachers(0),
        numTeachers(0),
        conferences(0),
        numConferences(0),
        assignmentOrder(0)
    {}
    ~TestCase()
    {
        delete [] conferences;
        delete [] teachers;
        delete [] assignmentOrder;
    }
    void Run()
    {
        MakeAssignmentOrder();
        LinkParentsWithChildren();
        SetupTeachers();
        CombineRequests();
        numRequests -=
        NumAbsorbedRequests();

        AllocateConferences();
    }
};

```

```

PopulateConferences();

ReduceWaits();
SendConferences();
}

private:
    void MakeAssignmentOrder()
// Generates a period order, from the least available to the most available,
// based on parent availability. Less available periods will be assigned first.
    {
        #if MAXNUMPERIODS > 255
            AvailabilityByPeriod* ABP;
            try {
                ABP=new AvailabilityByPeriod[1+numPeriods];
            } catch(...) {
                Error("Could not allocate storage for","ABP");
            }
        #else
            AvailabilityByPeriod ABP[256];
        #endif
        for (int period=0;period<=numPeriods;period++)
            ABP[period].Init(period);

        for (Parent* P=parents;P<parents+numParents;P++)
        {
            long firstPeriod=P->FirstPeriod();
            assert(firstPeriod>0);
            long lastPeriod=P->LastPeriod();

            assert(lastPeriod<=numPeriods);
            for (long
                period=firstPeriod;period<=lastPeriod;period++)
            {
                ABP[period].Increment();
            }
        }

        qsort(ABP+1,numPeriods,sizeof(AvailabilityByPeriod),
            Cmp<AvailabilityByPeriod>);

        try {
            assignmentOrder=new Period_t[1+numPeriods];
        } catch(...) {
            Error("Could not allocate storage
                for","assignmentOrder");
        }

        for (int period=0;period<=numPeriods;period++)
            assignmentOrder[period]=ABP[period].Period();

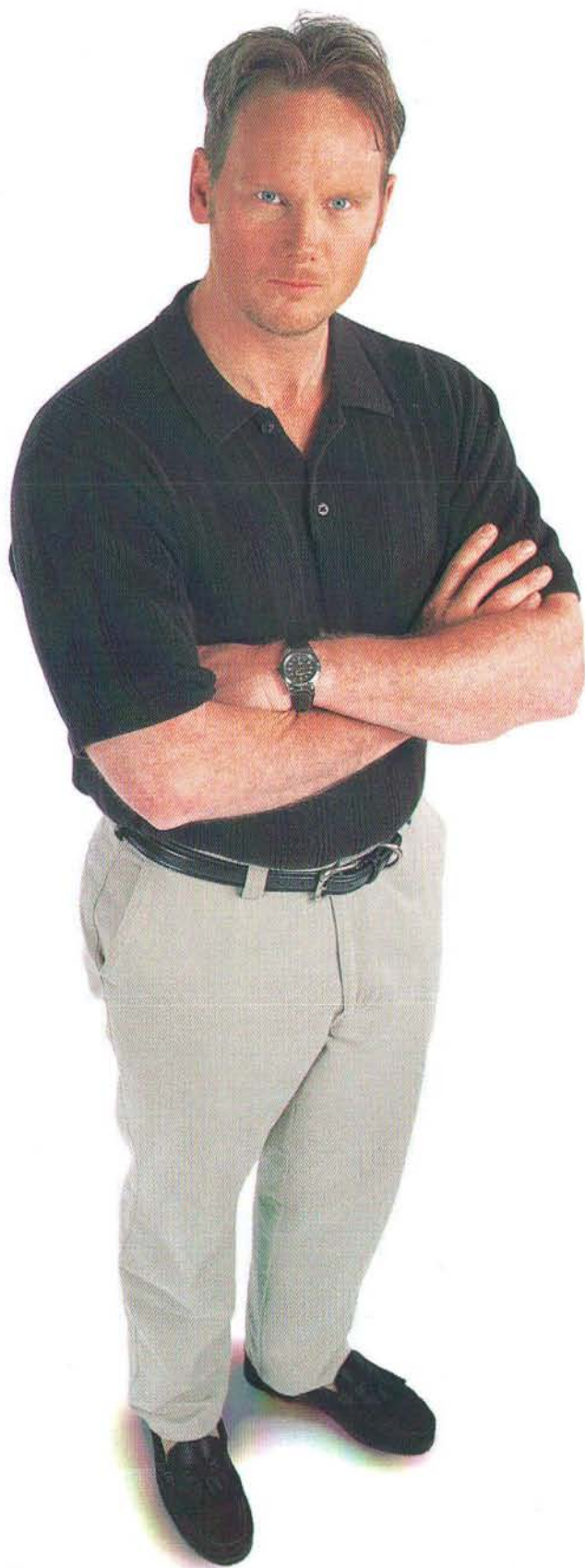
        #if MAXNUMPERIODS > 255
            delete [] ABP;
        #endif
    }

    void LinkParentsWithChildren()
// Replaces parentName and childName with parent and child pointers.
    {
        Child* C=children;
        for (long i=0;i<numChildren;i++,C++)
        {
            Parent* parent1=FindParent(C->Parent1Name());
            Parent* parent2=FindParent(C->Parent2Name());

            C->RegisterParents(parent1,parent2);
        }
        for (long i=0;i<numChildren;i++)
        {
            children[i].MarryParents();
        }
    }

    void SetupTeachers()
// Scans the already sorted requests (by teacher's name, from teachers file) to:
// discover unique teacher names, and setup in the teachers array,
// replaces childName with child (assumes childrenFile exists),
// installs availability in parents (assumes parents are linked to children)
// Then scans the requests again to make the teachers array
    {
        char* lastTeacherName="";
        char* teacherName=0;
    }
};

```

My files are
important.

Very important.

I only share my
files with those
I trust.

I trust DAVE.

Mac to PC, PC to Mac. Cross-platform file and print sharing is too vital to your business to risk. Trust Thursby, the company with 15 years experience. Trust DAVE, the solution with a proven track record. Share files and printers across a network with no barriers. DAVE installs on your Mac with no additional software required for the PC. It's fast, secure and easy to use. Download a free evaluation today!

Trust **DAVE**.



**New! Support
for inkjet
printing**



www.thursby.com

Mac is a trademark of Apple Computer, Inc. registered in the U.S. and other countries. The "Built for Mac OS X" graphic is a trademark of Apple Computer, Inc., used under license. DAVE is a registered trademark of Thursby Software Systems, Inc. © 2002, Thursby Software Systems, Inc.



and print
The File Share Folks™




```

Request* R=requests;
long teacherId=1;
for (long i=0;i<numRequests;i++,R++)
{
    teacherName=R->Name();
    if (strcmp(lastTeacherName,teacherName))
    {
        lastTeacherName=teacherName;
        teacherId++;
    }
    R->SetTempTeacherId(teacherId);

    char* childName=R->ChildName();
    Child* child=FindChild(childName);
    assert(child);
    R->ReplaceChildName(child);
    child->SetParentAvailability();
}

numTeachers=teacherId+1;
try {
    teachers=new Teacher[numTeachers];
} catch(...) {
    Error("Could not allocate storage for","teachers");
}

long lastTid=-1;
R=requests;
for (long i=0;i<numRequests;i++,R++)
{
    long tid=R->TempTeacherId();
    assert(tid<numTeachers);
    if (tid != lastTid)
    {
        teachers[tid].Init(R->Name(),tid);
        lastTid=tid;
    }
    R->SetTeacher(teachers[tid]);
}

void CombineRequests()
// Entered with requests already sorted by teacher.
// For each teacher's block of request, sorts by parent,
// Combines equivalent requests (same teacher, same parents, sums priorities)
// All absorbed requests are marked with negative priority
{
    Request* src=requests; // read from
    Request* end=numRequests+requests;
    for (int i=0;i<numTeachers;i++)
    {
        long
        numSameTeacher=CountByTeacher(src,end,&teachers[i]);
        qsort(src,numSameTeacher,sizeof(Request),Request::CmpParents);

        Request* uniqueRequest=src;
        for (Request* R=src+1;R<src+numSameTeacher;R++)
        {
            if (Request::CmpParents(R,uniqueRequest)) // if they differ
            {
                uniqueRequest=R;
            } else
            {
                R->TransferPriorityTo(uniqueRequest);
            }
        }
        src+=numSameTeacher;
    }
}

long NumAbsorbedRequests()
// Sorts requests by priority and counts the tail of -1 priority (nonUnique) requests
// returns the number of requests that can be dropped
{
    qsort(requests,numRequests,sizeof(Request),
        Request::CmpPriorities);
    Request* R=requests+numRequests;
    long numDropped=0;
    while ((R>requests) && (R->Priority() < 0))
        numDropped++;
}

return numDropped;
}

void AllocateConferences()
// There can not be more than numTeachers*numPeriods conferences
// unless one were to allow multiple parents to attend the same conference.
// This version does not support multiple conferences at the same time.
{
    numConferences=1+numTeachers*numPeriods; // 1 extra for [0] location
    try {
        conferences=new Conference[numConferences];
    } catch(...) {
        Error("Could not allocate storage for","conferences");
    }
    // address as conferences[teacherNr*numPeriods + periodNumber]
    // block pitch is numPeriods; conferences[0] is not used.
}

long PopulateConferences()
// Simple first come, first served, lower availability and higher priorities first.
// After assigning requests to conferences, the request array must not change.
// Returns the simple score (sum of priorities of requests accommodated).
{
    long numToSort=(numConferences-1 < numRequests)?
        (numConferences-1):numRequests;
    qsort(requests,numToSort,sizeof(Request),
        Request::CmpAvailability);

    Request* R=requests;
    long score=0;
    long numAssigned=0;
    for (int i=0;i<numRequests;i++,R++)
    {
        Teacher* teacher=R->TheTeacher();
        long teacherId=teacher->Pid();
        long priority=R->Priority();

        Conference* block=conferences+teacherId*numPeriods;

        // find first free conference
        for (int periodIndex=1;periodIndex<=numPeriods;
            periodIndex++)
        {
            long period=assignmentOrder[periodIndex];
            if (block[period].IsAssigned())
                continue;

            Parent* parent=R->GetAvailableParent(period);

            if (parent)
            {
                block[period].Init(teacher,parent,period,priority);
                parent->Book(block+period,period);
                score+=priority;
                numAssigned++;
                R->SetPeriod(period);
                break;
            }
        }
        if (numAssigned >= numConferences-1)
            break;
    }
    return score;
}

void ReduceWaits()
// Tries easy ways of reducing waiting times:
// moving to a different time (agreeable to both parent and teacher);
// transferring a conference to a spouse;
// swapping time slots with another parent (same teacher).
// These operations affect each other and are repeated until no more
// improvement can be obtained.
// ADDED in version 2:
// cancelling a scheduled conference if that generates a points advantage.
{
    long accumulatedReduction;
    do {

```




REAL Software and MacTech present the REALbasic Showcase to highlight some of the fantastic solutions created by REALbasic users worldwide. The showcase illustrates the wide range of applications that developers using REALbasic can create. Some benefit any Mac user, and others are more specific. All of them are seriously cool!

REALbasic is the powerful, easy-to-use tool for creating your own software for Macintosh, Mac OS X, and Windows. It runs natively on Mac OS X as well as earlier versions of the Mac OS. For more information, please visit: <www.realbasic.com>.

The Made with REALbasic program is a cooperative effort between REALbasic users and REAL Software, Inc. to promote the products created using REALbasic and the people who create them. For more information about the Made with REALbasic program, please visit: <www.realbasic.com/realbasic/mwrb/Partners/MwRbProgram.html>.

Extend REALbasic with Advanced Plugins

Spreadsheet Controls:

We provide a wide range of spreadsheet controls for REALbasic, including multistyled and custom rendering spreadsheet controls.

	A	B	C
	This is some incredible list		
1	Some text	More text	
2	Some text	More text	
3	Some text	More text	
4	Some text	More text	
5	Some text	More text	
6	Some text	More text	
7	Some text	More text	
8	Some text	More text	

- More than 32k of rows.
- Classic, OS X and Win32.
- Accelerated for maximum speed.
- Images in cells.



Cryptography:

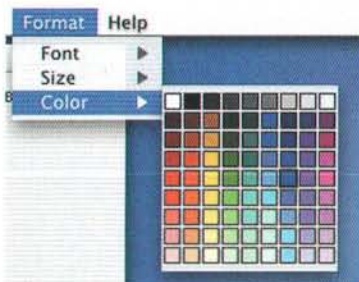
We provide data encryption, encoding, compression and hashing for REALbasic.

Supported Algorithms:

- Encryption:
 - e-CryptIt
 - BlowFish (448 bit)
 - AES (Rijndael) (256 bit)
- Encoding:
 - e-CryptIt Flexible
 - Base 64
 - BinHex
 - MacBinary III
 - AppleSingle / Double
 - UUcoding
- Compression:
 - Zip on Strings and streams (.gz)
- Hashing and Checksums:
 - CRC32 / Adler32
 - MD5 / HMAC_MD5
 - SHA / SHA1 / HMAC_SHA1

Other Plugins:

We have many other plugins for REALbasic, including plugins to do advanced MacOS Toolbox tasks and more custom Controls.



Speed up development and make more advanced applications by using plugins ! Get free demos at www.einhugur.com



Einhugur Software
sales@einhugur.com
www.einhugur.com

Corona

accounting software



"Easy to set up, easy to use, and excellent support from the developer."

Five stars on VersionTracker.com

- cash entry
- invoicing
- payroll
- reports

- account chart
- address index
- help facility

\$49.95

Free 30-day trial

<http://homepage.mac.com/idlewild/CoronaUS.hqx>



A best friend for business!

p.o. box 3164 • newberg • oregon • 97132
idlewild@mac.com

crescendo software



Picture Play

Create digital collages on your Mac, quickly and easily.

Combine digital photos, clipart, or anything else to make a work of art in no time!
 Picture Play is intuitive and powerful enough to be used by itself,
 but not so complex that first time users will be bewildered.
 Great value, it will let you make wonderful compositions more simply than ever.

Only US\$10, for Mac OS 9 and Mac OS X.

Check it out by going to <http://www.crescendosw.com/> today!

Your product can go here!



Advertising helps you build your brand. After all, you don't expect someone to buy your product if they've never heard of you, do you?

All products/services made with or related to REALbasic qualify to be a part of our exclusive marketplace section.

Are you looking for extremely small out of pocket advertising that will reach your target audience?

We bill you for ads monthly as each issue ships.

Contact our ad sales team today for more details.

Voice: 805-494-9797 x113 • eMail: adsales@mactech.com

MacTech
www.mactech.com



iScreensaver Designer

for Macintosh and Windows



Mac designers...
build professional screensavers
without touching a Windows machine

the world's only cross-platform screensaver design tool

- build both Macintosh and Windows screensavers with a single click, on whatever system* you are using!
- use any QuickTime movie format :
Macromedia Flash, MPEG, Cinepak, MP3, Midi, AVI, DV Video...
- include a hidden movie that can be unlocked with a registration code
- customize and fully-brand both Installers and Screensaver control panels with pictures and text
- screensavers install without DLLs, extensions, or restarts

simple
WYSIWYG
editor

supports
interactive Flash
and QuickTime

consistent
cross-platform
user interface

try before you buy
fully functional
online downloads



the iScreensaver Designer editing environment

<http://iScreensaver.net>
email : info @ iScreensaver.net

* supported systems, as of November 2001, include:
Macintosh OS 8.6 to 9.2, Microsoft Windows 95/98/ME, NT4/NT2000/XP

©2001 Kichi Media Inc. All Rights Reserved.
iScreensaver Designer and the Doublet logo are trademarks of Kichi Media Inc.
Apple, Macintosh, and QuickTime are trademarks and/or registered trademarks of Apple Computer, Inc.
Macromedia Flash is a trademark of Macromedia, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation.
All other trademarks are the property of their respective owners. Made using REALbasic.

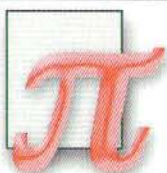
CREATIVE[BOX]

EAT. SLEEP. DESIGN.

FontViewer

FontViewer is a simple but elegant font viewer that allows you to preview all fonts in your fonts folder. They can be displayed in various sizes and styles with the click of a mouse. FontViewer also features the ability to display your fonts in a slideshow, and view your fonts in almost every type of environment possible (i.e. desktop, menus, icons, etc.)

Creative Box URL: <http://www.creativebox.net/>
Developer e-mail: jason@creativebox.net



piDock

piDock is the must have navigation and launching utility for MacOS. Now you can easily organize, navigate and launch files with a simple sweep of the mouse.

Compatible with OS 8.6 through 10.1, OSX users can now control-option click a folder to navigate it's contents.

<http://www.pidog.com> • <http://www.pidock.com>
piDock@pidog.com



StimpSoft™, Inc.

StimpSoft™, Inc. Makers of insanely mediocre software for cats, dogs, voles, wallabies, mangos, crickets, and people. We put the "L" in "Software".

URL: <http://www.stimpsoft.com>
Email: john@stimpsoft.com



abDataTools

db Reports is a cross-platform report writing application able to extract, format and print data from one of various data sources. A multitude of built-in features and expressions make db Reports a powerful tool for analyzing data.

Free demo and more details located at
<http://abDataTools.com>

Whistle Blower

Enterprise server monitor and restart utility
whistleblower.sentman.com

Connect to and validate the response from web servers, cgi scripts and over 23 other types of servers.

Send email, pages and perform unattended restarts via MasterSwitch or PowerKey.

Shifts make sure that the person on call when the server goes down is the one who gets the page.

68k, PPC and Carbon

Web based administration lets you check on and restart your servers from anywhere.

Customize your response to an outage with Apple Script.

email us at whistleblower@sentman.com

TitleTrack™ JUKEBOX

MAC MUSIC MANAGEMENT FOR
SONY® 5 TO 400 DISC CD CHANGERS



Control up to 12 changers (4,800 music CDs)
Control Any Brand Stereo Receiver
Play MP3 and other Sound Files
Plus much, much more!!!

www.titletrack.com

info@titletrack.com

REALbasic® For Dummies®

Erick Tejkowski's guide for beginning REALbasic while creating useful and impressive applications is a 'must have' in any REALbasic reference library.

This book includes a professional reference that will give you the functionality you need in a hurry.

Go to:

www.devdepot.com/realbasicfordummies
and buy today for over \$8 off retail price!

Available While Supplies Last!

DEV DEPOT
www.devdepot.com

Unlock the power of OS X!



MONTAGE
Technologies Incorporated

Mac OS X and WebObjects Development and Consulting Services
1-866-245-8485 www.montagetech.com


```

    accumulatedReduction = ReduceWaitsByTimeChange();

    long reduction;
    while (0 < (reduction =
ReduceParentWaitsByTransfers()))
        accumulatedReduction+=reduction;

    accumulatedReduction += ReduceParentWaitsBySwapping();
    while (accumulatedReduction>0);

    ReducePenaltyByCancellation();
}

Child* FindChild(char* childName)
// Finds a child record by name.
{
    Child searchKey(childName,0,0,0);
    return (Child*)
    bsearch(&searchKey,children,numChildren,sizeof(Child),
        Cmp<Child>);
}

Parent* FindParent(char* parentName)
// Finds a parent record by name.
{
    if (parentName)
    {
        Parent searchKey(parentName,0,0,0);
        return (Parent*)
        bsearch(&searchKey,parents,numParents,sizeof(Parent),
            Cmp<Parent>);
    } else return 0;
}

long CountByTeacher(Request* R,Request* end,Teacher*
teacher)
// Returns the number of requests for one teacher.
{
    long count=0;
    while ((R<end) && ((R++)->TheTeacher()==teacher))
        count++;
    return count;
}

long ReduceParentWaitsBySwapping()
// Attempts to swap conferences with the same teacher, but between different parents,
// if that results in a reduction of waiting penalties.
// Returns the reduction achieved.
{
    long reduction=0;
    for (Conference* block=conferences;
        block<conferences+numConferences-1;block+=numPeriods)
    {
        for (long period1=1;period1<numPeriods;period1++)
        {
            Conference* C1=block+period1;
            for (long
period2=period1+1;period2<=numPeriods;period2++)
            {
                Parent* P1=C1->TheParent();
                if (!P1)
                    continue;
                if (P1->IsBooked(period2) || !P1-
>IsAvailable(period2))
                    continue;
                Conference* C2=block+period2;
                Parent* P2=C2->TheParent();
                if (!P2)
                    continue;
                if (P2->IsBooked(period1) || !P2-
>IsAvailable(period1))
                    continue;
                long before=P1->NumIdlePeriods()+P2-
>NumIdlePeriods();
                if (before<=0)
                    continue;
                P1->SwapConference(period1,P2,period2);
                long after=P1->NumIdlePeriods()+P2-
>NumIdlePeriods();
                if (after <= before)
                {
                    C1->UpdateParent(P2);
                    C2->UpdateParent(P1);

                    reduction += before-after;
                    else // swap back
                    P1->SwapConference(period2,P2,period1);
                }
            }
        }
        return reduction;
    }

    long ReduceParentWaitsByTransfers()
// Attempts to reduce waiting penalties by assigning conferences to spouses.
// Returns the reduction achieved.
{
    long reduction=0;
    for (Conference* C=conferences;
        C<conferences+numConferences;C++)
    {
        Parent* P=C->TheParent();
        if (!P)
            continue;
        Parent* spouse=P->Spouse();
        if (!spouse)
            continue;
        long period=C->Period();
        if (!spouse->IsAvailable(period) ||
            spouse->IsBooked(period))
            continue;
        long before=P->NumIdlePeriods() + spouse-
>NumIdlePeriods();
        if ((before<=0))
            continue;
        P->TransferConferenceToSpouse(period);
        long after=P->NumIdlePeriods() + spouse-
>NumIdlePeriods();

        if (after <= before) // update conference with transfer
        {
            C->UpdateParent(spouse);
            reduction+=before-after;
        } else
            spouse->TransferConferenceToSpouse(period); // undo
    }
    transfer
    {
        return reduction;
    }

    long ReduceWaitsByTimeChange()
// Attempts to reduce waiting penalties by shifting conferences to different periods.
// Returns the reduction achieved.
{
    long reduction=0;
    for (Conference* C=conferences;
        C<conferences+numConferences;C++)
    {
        Parent* P=C->TheParent();
        if (!P)
            continue;
        long bestPeriod=C->Period();
        Teacher* T=C->TheTeacher();
        assert(T);
        Conference* teachersBlock=conferences+T-
>Pid()*numPeriods;
        long lowestWait=P->NumIdlePeriods()+TeacherWaits(T);

        if (lowestWait<=0)
            continue;
        long before=lowestWait;
        for (int alternatePeriod=1;alternatePeriod<=numPeriods;
            alternatePeriod++)
        {
            if (alternatePeriod==bestPeriod)
                continue;
            if (P->IsAvailable(alternatePeriod) &&
                !P->IsBooked(alternatePeriod) &&
                !teachersBlock[alternatePeriod].IsAssigned())
            {
                P-
>RescheduleConference(bestPeriod,alternatePeriod);
                teachersBlock[alternatePeriod].
                Copy(teachersBlock[bestPeriod],alternatePeriod);
                teachersBlock[bestPeriod].Clear();
                long wait=P->NumIdlePeriods()+TeacherWaits(T);
                if (wait<=lowestWait)
                {

```


THE EASY, COMPACT WAY TO SOLVE PROBLEMS!

MacTech[®] CD-ROM

www.mactech.com



The MacTech CD-ROM with THINK Reference is the essential reference resource for Macintosh programmers. This CD includes the THINK Reference personal database system and a wealth of Macintosh programming databases, featuring almost 200 issues of the journal of Macintosh programming – MacTech Magazine. This release also features the THINK Reference Compiler, which allows you to compile HTML files into your own compact, searchable THINK Reference databases.

PO Box 5200 • Westlake Village, CA • 91359-5200
800/MACDEV-1 (800/622-3381) • Outside US/Canada: 805/494-9797
Fax: 805/494-9798 • E-mail: orders@devdepot.com


```

        lowestWait=wait;
        bestPeriod=alternatePeriod;
    } else // undo the rescheduling
    {
        P-
>RescheduleConference(alternatePeriod,bestPeriod):
        teachersBlock[bestPeriod].

Copy(teachersBlock[alternatePeriod],bestPeriod);
        teachersBlock[alternatePeriod].Clear();
    }
}
    reduction += before-lowestWait;
}
return reduction;
}

long ReducePenaltyByCancellation()
// Attempts to reduce waiting penalties by cancelling conferences entirely.
// Returns the reduction achieved.
{
    long reduction=0;
    for (Conference* C=conferences+1;
        C<conferences+numConferences;C++)
    {
        if (!C->IsAssigned()) continue;
        Parent* P=C->TheParent();
        Teacher* T=C->TheTeacher();
        long priority=C->Priority();
        long waitBefore=P->NumIdlePeriods()+TeacherWaits(T);
        long period=C->Period();
        P->Unbook(period); // cancel conference
        C->Clear();
        long waitAfter=P->NumIdlePeriods()+TeacherWaits(T);
        if (waitAfter < waitBefore-priority)
        {
            reduction += waitBefore-priority - waitAfter;
        } else // reestablish conference
        {
            P->Book(C,period);
            C->Init(T,P,period,priority);
        }
    }
    return reduction;
}

void SendConferences()
// Sends the conference schedule to the output file.
{
    outputFile.Open();
    for (int i=0;i<numConferences;i++)
    {
        if (conferences[i].IsAssigned())
        {
            outputFile.SendConference(conferences[i]);
        }
    }
    // Version 3:
    // Let spouse join a conference if that reduces waiting time for the spouse.
    Parent* spouse=conferences[i].TheParent()->Spouse();
    if (spouse)
    {
        long period=conferences[i].Period();
        if (spouse->IsAvailable(period) &&
            !spouse->IsBooked(period) &&
            // make sure extra conference improves spouse's schedule
            (period < spouse->LastPeriodUsed()) &&
            (period > spouse->FirstPeriodUsed()))
        {
            // We just replace parent with spouse since parent's conference is already recorded.
            conferences[i].UpdateParent(spouse);
            outputFile.SendConference(conferences[i]);
        }
    }
}
outputFile.Close();

long TeacherWaits(Teacher* T)
// Assumes the conference array is laid out as rows of numPeriod conferences per
// teacher. Returns the number of idle periods of one teacher.
{
    long teacherId=T->Pid();

```

```

        long teacherfirst=numPeriods+1;
        long teacherLast=0;
        long teacherNum=0;
        Conference* block=conferences+teacherId*numPeriods;
        for (int period=1;period<=numPeriods;period++)
        {
            if (block[period].IsAssigned())
            {
                if (period<teacherfirst) teacherfirst=period;
                teacherLast=period;
                teacherNum++;
            }
        }
// FIXED in version 2: Correct output (0) if teacher is not assigned any conference
        long teacherWaits=(teacherNum) ?
            (teacherLast+1-teacherfirst-teacherNum):0;
        return teacherWaits;
    }
}

int main()
{
    long numTests=InputFile::GetNumTests(); // get the
    number of tests
    LogFile logFile(numTests);

    for (long testNumber=1;testNumber<=numTests;testNumber++)
        // loop over the number of tests
    {
        MyTimer timer; // starts a timer
        TestCase testCase(testNumber); // constructs a test case
        testCase.Run(); // runs the test case
        logFile.Set( // saves the timing in logFile
            testNumber,timer.ElapsedMilliSecs());
    }
    logFile.Send(); // sends the logFile to disk
    return 0;
}

```

PT-PERSONS.H

```

#ifndef PT_PERSONS_H
#define PT_PERSONS_H
/* Person based classes for the Parent-Teacher conferences */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "PT-compile-constants.h"
#include "PT-Utilities.h"

enum {
    kLineLength=255,
    kLineEnd=0x0D // Mac line end
};

class Request;
typedef Request* RequestPtr;

class Conference;
typedef Conference* ConferencePtr;

class Person
// Base class, knows how to scan the text to extract the persons name.
{
    char* name;
    ulong pid;
public:
    Person():name(0){}
    Person(char* xname,ulong xid):name(xname),pid(xid){}
    char* Name() const {return name;}
    ulong Pid() const {return pid;}

    int operator - (Person& r) const {return
    strcmp(name,r.name):;}

    char* Init(char* cp,long xid)
    {
        char* line=LeftTrim(cp);
        name=line;
    }
}

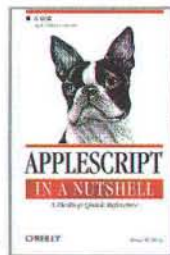
```


Trying to keep up with OS X?

WE CAN GET YOU ON THE INSIDE TRACK.

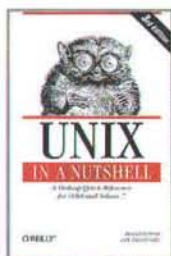
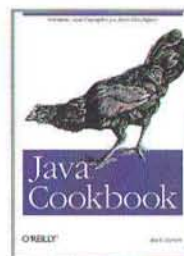
WE'RE TRUE INSIDERS.

O'Reilly and Apple have joined forces to bring you unparalleled access to the experts developing OS X. Get on the inside track (and stay there) with our OS X books and the O'Reilly Network's Mac DevCenter at www.oreillynet.com/mac.



WE KEEP PACE WITH JAVA.

Our Java books—and our online site for Java developers, on.java.com—offer the most comprehensive and up-to-date information available to help you keep pace, too.



WE KNOW UNIX—INSIDE AND OUT.

We cut our teeth on Unix: O'Reilly has been publishing the most respected and useful books on Unix for well over a decade, and we're still at the head of the pack.

O'REILLY®


```

    line=ForwardTo(line, '.');
    *line=0;
    RightTrim(name, line);
    pid=xid;
    return line;
}

static char* LeftTrim(char* cp)
{
    while (*cp && !isalnum(*cp))
        cp++;
    return cp;
}

static void RightTrim(char* start, char* cp)
// Scans backwards, beginning at the null char beyond string
{
    while ((cp>start) && !isalnum(*(cp-1)))
        *cp=0;
}

static char* ForwardTo(char* cp, char c)
{
    while (*cp && (*cp != c))
        cp++;
    return cp;
}

static char* ForwardTo(char* cp, char c1, char c2)
{
    while (*cp && (*cp != c1) && (*cp != c2))
        cp++;
    return cp;
}
};

#if MAXNUMPERIODS <= 255
typedef uchar Period_t;
#define Period_FORMAT "%hhd"
#else
typedef long Period_t;
#define Period_FORMAT "%ld"
#endif
class Parent:public Person
// Collection of parental functions.
{
    Parent* spouse;
    ConferencePtr* booked;
    // array[numPeriods] allocated if/when parent books a conference
    Period_t firstPeriod;
    Period_t lastPeriod;
    Period_t firstPeriodUsed;
    Period_t lastPeriodUsed;
    Period_t numPeriodsUsed;
public:
    Parent() :
        spouse(0),
        booked(0)
    {}
    Parent(char* parentName, long xid, long p1, long p2) :
// This constructor is only used in making a searchkey for bsearch
        Person(parentName, xid),
        spouse(0),
        booked(0),
        firstPeriod(p1),
        lastPeriod(p2),
        firstPeriodUsed(lastPeriod+1),
        lastPeriodUsed(0),
        // we'll rely on lastPeriodUsed initialized to 0 to indicate no-conference
        numPeriodsUsed(0)
    {}
    ~Parent() {if (booked) delete [] booked;}
    char* Init(char* cp, long xid)
// initializes each parent as part of reading the schedules file
    {
        char* line=Person::Init(cp, xid);
        sscanf(++line, Period_FORMAT, "Period_FORMAT", &firstPeriod,
            &lastPeriod);
        firstPeriodUsed=lastPeriod+1;
        lastPeriodUsed=0; // we'll rely later on this having been initialized to 0
        numPeriodsUsed=0;
        line=ForwardTo(line, kLineEnd);
        return line+1;
    }

    Parent* Spouse() const {return spouse;}
    void Marries(Parent* s) {spouse=s;}
    const Parent* FirstParent() const // returns parent of higher address
    {
        if (this > spouse)
            return this;
        else
            return spouse;
    }

    long FirstPeriod() const {return firstPeriod;}
    long LastPeriod() const {return lastPeriod;}
    long FirstPeriodUsed() const {return firstPeriodUsed;}
    long LastPeriodUsed() const {return lastPeriodUsed;}
    bool IsAvailable(long period) const
    {
        return ((period>=firstPeriod) && (period<=lastPeriod));
    }

    void Book(Conference* C, long period)
    {
        numPeriodsUsed++;
        if (period>lastPeriodUsed) lastPeriodUsed=period;
        if (period<firstPeriodUsed) firstPeriodUsed=period;
        if (!booked)
        {
            booked=new ConferencePtr[1+lastPeriod];
            memset(booked, 0, sizeof(ConferencePtr)*(1+lastPeriod));
        }
        booked[period]=C;
    }

    ConferencePtr IsBooked(Period_t period) const
    {
        if (period>lastPeriodUsed) return 0;
        return booked[period];
    }

    void TransferConferenceToSpouse(long period)
    {
        assert(IsBooked(period));
        assert(spouse);
        assert(!spouse->IsBooked(period));
        Conference* C=booked[period];
        Unbook(period);
        spouse->Book(C, period);
    }

    void SwapConference(Period_t period1, Parent* P2,
        Period_t period2)
    {
        Conference* C1=booked[period1];
        assert(C1);
        Unbook(period1);
        Conference* C2=P2->booked[period2];
        assert(C2);
        P2->Unbook(period2);
        Book(C2, period2);
        P2->Book(C1, period1);
    }

    void Unbook(long period)
    {
        booked[period]=0;
        if (period==firstPeriodUsed)
        {
            if (period==lastPeriodUsed)
            {
                firstPeriodUsed=lastPeriod+1;
                lastPeriodUsed=0; // reinit
            }
            else
            {
                while (!booked[++firstPeriodUsed])
                    assert(firstPeriodUsed<lastPeriod);
            }
        }
        else if (period==lastPeriodUsed)
        {
            while (!booked[--lastPeriodUsed])
                assert(lastPeriodUsed>firstPeriod);
        }
        numPeriodsUsed--;
    }

    void RescheduleConference(long oldPeriod, long newPeriod)
    {
        Conference* C=booked[oldPeriod];
        assert(C);
        Unbook(oldPeriod);

```



```

    assert(!IsBooked(newPeriod));
    Book(C.newPeriod);
}

long NumIdlePeriods() const
{
    long numIdle=0;
    if (lastPeriodUsed)
        numIdle=1+lastPeriodUsed-firstPeriodUsed-
numPeriodsUsed;

    return numIdle;
}

void Show()
{
    printf("%s. %d. %d\n",Name(),firstPeriod,lastPeriod);
}

};

class Teacher:public Person
// Collection of teachers functions.
// In this program, teachers have no functions, and are just persons.
{
};

class Child:public Person
// Converts parent names into pointers to parent records.
// Collection of child functions, mostly about their parents though.
{
    union
    {
        char* parent1Name;
        Parent* parent1;
    };
    union
    {
        char* parent2Name;
        Parent* parent2;
    };
    Period_t parentAvailability;
public:
    Child():parent1(0),parent2(0){}
    Child(char* childName,long xid,char* p1,char* p2) :
        Person(childName,xid),
        parent1Name(p1),
        parent2Name(p2),
        parentAvailability(0)
    {}
    char* Init(char* cp,long xid)
    {
        char* line=Person::Init(cp,xid);

        line=LeftTrim(line+1);
        parent1Name=line;
        line=ForwardTo(line,'.',kLineEnd);
        bool secondParent=(*line==' ');
        *line=0;
        RightTrim(parent1Name,line);

        if (secondParent)
        {
            line=LeftTrim(line+1);
            parent2Name=line;
            line=ForwardTo(line,kLineEnd);
            *line=0;
            RightTrim(parent2Name,line);
        }
        else parent2Name=0;
        parentAvailability=0;// not known until parents are linked
        return line+1;
    }
    char* Parent1Name() const {return parent1Name;}
    char* Parent2Name() const {return parent2Name;}

    Parent* Parent1() const {return parent1;}
    Parent* Parent2() const {return parent2;}

    void RegisterParents(Parent* p1,Parent* p2)
    {
        assert(!strcmp(p1->Name(),parent1Name));
        parent1=p1;
        if (p2)
        {
            assert(!strcmp(p2->Name(),parent2Name));
            parent2=p2;

```

```

        }
    }
}

void MarryParents()
{
    assert(parent1);
    Parent* spouse1=parent1->Spouse();
    if (parent2)
    {
        Parent* spouse2=parent2->Spouse();
        if (!spouse1 && !spouse2)
        {
            parent1->Marries(parent2);
            parent2->Marries(parent1);
        }
    }
}

long ParentAvailability() const {return
parentAvailability;}
void SetParentAvailability()
// because of possibly mixed parent sets, this needs to be computed for each child
{
    parentAvailability=ComputeParentAvailability();
}

long ComputeParentAvailability()
// returns total number of periods either parent is available
{
    long p1first=parent1->FirstPeriod();
    long p1last=parent1->LastPeriod();
    long p1length=1+p1last-p1first;
    if (parent2)
    {
        long p2first=parent2->FirstPeriod();
        long p2last=parent2->LastPeriod();
        long p2length=1+p2last-p2first;
        if (p1first<=p2first)
        {
            if (p2first > p1last) return p1length+p2length;
            // disjoint or contiguous
            if (p2first <= p1last)
                if (p2last>p1last) return 1+p2last-p1first;
            else return p1length;
        }
        else // p2first < p1first
        {
            if (p1first > p2last) return p1length+p2length;
            // disjoint or contiguous
            if (p1first <= p2last)
                if (p1last>p2last) return 1+p1last-p2first;
            else return p2length;
        }
    }
    return p1length;
}

};

class Request:public Person
// Teachers name from file is stored in Person.
// Expansion of the data gathered from the teachers file.
{
    ulong tempTeacherId;
    Teacher* teacher;
    union
    {
        char* childName;
        Child* child;
    };
    Period_t assignedPeriod;
    long priority;
public:
    // We rely on memset() to clear all to 0 when an array of Requests is allocated.
    char* Init(char* cp,long xid)
    {
        char* line=LeftTrim(cp);
        childName=line;
        line=ForwardTo(line,'.','.');
        *line=0;
        RightTrim(childName,line);

        line=Person::Init(line+1,xid);
        sscanf(++line,"%ld",&priority);
        line=ForwardTo(line,kLineEnd);
        return line+1;
    }
    long Priority() const {return priority;}
    void ReplaceChildName(Child* c)

```



```

    assert(!strcmp(c->Name(),childName));
    child=c;
}
ulong TempTeacherId() const {return tempTeacherId;}
void SetTempTeacherId(ulong tid) {tempTeacherId=tid;}

Teacher* TheTeacher() const {return teacher;}
char* TeacherName() const {return teacher->Name();}
void SetTeacher(Teacher* theTeacher){teacher=theTeacher;}

char* ChildName() const {return childName;}
Child* TheChild() const {return child;}

Parent* Parent1() const {return child->Parent1();}
Parent* Parent2() const {return child->Parent2();}
Parent* GetAvailableParent(long period) const
{
    Parent* parent=Parent1();
    if (parent->IsAvailable(period) && !parent->IsBooked(period))
        return parent;
    parent=Parent2();
    if (parent && parent->IsAvailable(period) && !parent->IsBooked(period))
        return parent;
    return 0;
}

long ParentAvailability() // precomputed for all parent pairs ahead of time
// returns number of periods either parent is available
{
    return child->ParentAvailability();
}

void TransferPriorityTo(Request* uniqueRequest)
{
    uniqueRequest->priority += priority;
    priority-=1;
}

long IsAssigned() const {return assignedPeriod;}
void SetPeriod(Period_t period) {assignedPeriod=period;}

static int CmpParents(const void* a,const void* b)
// compares parents simply by their address in memory
{
    Request* ra=(Request*)a;
    Request* rb=(Request*)b;
    int d=ra->Parent1()-rb->Parent1();
    if (d==0)
        return ra->Parent2()-rb->Parent2();
    return d;
}

static int CmpPriorities(const void* a,const void* b)
// highest priority first
{
    Request* ra=(Request*)a;
    Request* rb=(Request*)b;
    int d=rb->priority-ra->priority;
    return d;
}

static int CmpAvailability(const void* a,const void* b)
// lowest availability first, then highest priority
{
    Request* ra=(Request*)a;
    Request* rb=(Request*)b;
    int d=ra->ParentAvailability()-rb->ParentAvailability();
    if (d==0)
    {
        d=rb->priority-ra->priority;
    }
    return d;
}

};

class Conference
// Holds data defining a conference.
// Collection of conference related functions.
{
    Teacher* teacher;
    Parent* parent;
    Period_t period;
    long priority;

```

```

public:
    Teacher* IsAssigned() const {return teacher;}
    void Init(Teacher* t,Parent* p,Period_t x,long y)
    {
        teacher=t;
        parent=p;
        period=x;
        priority=y;
    }
    void Clear() {Init(0,0,0,0);}
    void Copy(Conference& C,Period_t x)
    {
        teacher=C.teacher;
        parent=C.parent;
        period=x;
        priority=C.priority;
    }
    Teacher* TheTeacher() const {return teacher;}
    Parent* TheParent() const {return parent;}
    const Parent* FirstParent() const
    {
        assert(parent);
        return parent->FirstParent();
    }
    long Period() const {return period;}
    long Priority() const {return priority;}
    void UpdateParent(Parent* p) {parent=p;}
    int operator - (Conference& r) const
    {
        int d=r.parent->FirstParent()-parent->FirstParent();
        if (d==0)
            return period-r.period;
        return d;
    }
};

#endif

```

PT-UTILITIES.CPP

```

/* Some basic utilities.
   Mostly self-explanatory.
*/
#include <string.h>
#include "PT-Utilities.h"

void Error(char* s1,char* s2)
{
    printf("%s %s\n",s1,s2);
    exit(0);
}

void Error(char* s1,long x,char* s2)
{
    printf("%s%d%s\n",s1,x,s2);
    exit(0);
}

long FileSize(FILE* f)
{
    long curPos=ftell(f);
    fseek(f,0,SEEK_END);
    long size=ftell(f);
    fseek(f,curPos,SEEK_SET);
    return size;
}

```

PT-UTILITIES.H

```

#ifndef PT_UTILITIES_H
#define PT_UTILITIES_H

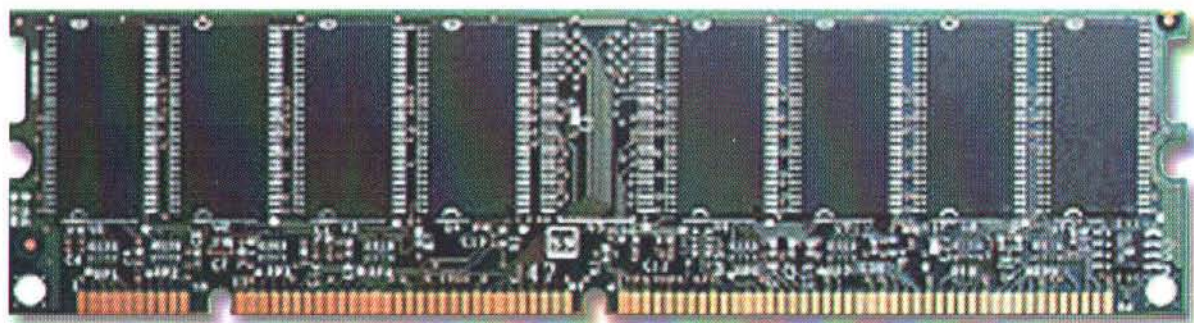
/*
   Prototypes of utilities, and a generic comparator.
   Mostly self-explanatory.
*/

#include <stdio.h>
#include <stdlib.h>

typedef unsigned char uchar;

```


There are some things in life you
can never have enough of...



Built-to-order. Lifetime Guarantee. Competitive Pricing.

**DEV
DEPOTSM**

www.devdepot.com/memory

Voice: 877-DEPOT-NOW (877-337-6866) • Outside US/Canada: 805/494-9797
E-mail: orders@devdepot.com


```

typedef unsigned long ulong;
typedef unsigned short ushort;

void Error(char* s1,char* s2);
void Error(char* s1,long x,char* s2);

long FileSize(FILE* f);

template <class T>
inline int Cmp(const void* a,const void* b)
// Generic comparator for qsort and bsearch
// Uses operator - of <T> for the actual comparison
{
    return *((T*)a) - *((T*)b);
}

#endif

```

PT-FILES.H

```

#ifndef PT_FILES_H
#define PT_FILES_H

/*
    File classes.
    Mostly self-explanatory.
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "PT-compile-constants.h"
#include "PT-Persons.h"

static char inputFileNames[] = INPUT_DIRECTORY"input.txt";
static char childrenFileName[] = INPUT_DIRECTORY"childrenNN.txt";
static char teachersFileName[] = INPUT_DIRECTORY"teachersNN.txt";
static char schedulesFileName[] = INPUT_DIRECTORY"schedulesNN.txt";
static char outputFileNames[] = INPUT_DIRECTORY"conferencesNN.txt";
static char logFileName[] = INPUT_DIRECTORY"log.txt";

enum {kOutputFileBufferSize=1024};

inline void SetFilename(char* name,unsigned long number)
// Replaces NN with a 2-digit number in a file name
{
    assert(number<=99);
    int len=strlen(name);
    assert(len>=6);
    char NN[4];
    sprintf(NN,"%02d",number);
    name[len-6]=NN[0];
    name[len-5]=NN[1];
}

class InputFile
// Only used to read the number of tests.
{
public:
    static long GetNumTests()
    {
        FILE* f=fopen(inputFileName,"rb");
        if (!f) Error(inputFileName,"not found");
        char line[kLineLength+1];
        fgets(line,kLineLength,f);
        long numTests;
        sscanf(line,"%ld",&numTests);
        fclose(f);
        return numTests;
    }
};

class TextFile
// Generic ancestor for text files.
// Reads the entire file into memory.
// Derived classes are required to parse the text in memory.
{

```

```

    long capacity;
    char* fileStore;
public:
    TextFile() : fileStore(0) {}
    char* FileStore() const {return fileStore;}
    long Capacity() const {return capacity;}
    long Read(char* fileName)
// reads file into fileStore
// returns number of text lines read
    {
        FILE* f=fopen(fileName,"rb");
        assert(f);
        capacity=FileSize(f);
        try {
            fileStore=new char[1+capacity]; // add 1 for NULL char at end
        } catch(...) {
            Error("Could not allocate storage for",fileName);
        }
        long numRead=fread(fileStore,1,capacity,f);
        assert (capacity == numRead);
        fclose(f);
        fileStore[capacity]=0; // zero terminate
        return LineCount();
    }
    ~TextFile()
    {
        if (fileStore)
            delete [] fileStore;
        fileStore=0;
    }
private:
    long LineCount() // assumes single line end character (Mac)
    {
        char* cp=fileStore-1;
        char c=*cp;
        long count=0;
        while (c)
        {
            if (c==kLineEnd) count++;
            c=*(++cp);
        }
        c=*(-cp);
        if (c != kLineEnd) count++;
        return count;
    }
};

class SchedulesFile:public TextFile,public Person
// Reads the PT-schedules file and sorts parents by name.
// Person class contains the "ForwardTo" function required here.
{
    Parent* parents;
public:
    SchedulesFile(long testNumber,long& numPeriods,
        Parent* &xparents,long& numParents)
    {
        SetFilename(schedulesFileName,testNumber);
        numParents=Read(schedulesFileName)-1;
        try {
            parents=new Parent[numParents];
            xparents=parents;
        } catch(...) {
            Error("Could not allocate storage for","parents");
        }
        char* cp=FileStore();
        sscanf(cp,"%ld",&numPeriods);
        if (numPeriods>MAXNUMPERIODS)
        {
            Error(
                "The number of periods exceeds MAXNUMPERIODS = ",
                MAXNUMPERIODS,
                "\nPlease adjust MAXNUMPERIODS in PT-compile-
                constants.h, and recompile");
        }
        cp=1+ForwardTo(cp,kLineEnd);
        for (long i=0;i<numParents;i++)
        {
            assert(*cp);
            cp=parents[i].Init(cp,i);
        }
        qsort(parents,numParents,sizeof(Parent),Cmp<Parent>);
    }
    ~SchedulesFile(){delete [] parents;}
};

class ChildrenFile:public TextFile

```



```

// Reads the PT-children file and sorts children by name.
{
    Child* children;
public:
    ChildrenFile(long testNumber, Child* &xchildren,
                 long& numChildren)
    {
        SetFilename(childrenFileName, testNumber);
        numChildren=Read(childrenFileName);
        try {
            children=new Child[numChildren];
            xchildren=children;
        } catch(...) {
            Error("Could not allocate storage for", "children");
        }
        char* cp=FileStore();
        for (long i=0; i<numChildren; i++)
        {
            assert(*cp);

            cp=children[i].Init(cp, i);
        }
        qsort(children, numChildren, sizeof(Child), Cmp<Child>);
    }
    ~ChildrenFile() {delete [] children;}
};

class TeachersFile:public TextFile
// Reads the PT-teachers file.
{
    Request* requests;
public:
    TeachersFile(long testNumber, Request* &xrequests,
                 long& numRequests)
    {
        SetFilename(teachersFileName, testNumber);
        numRequests=Read(teachersFileName);
        try {
            requests=new Request[numRequests];
            memset(requests, 0, sizeof(Request)*numRequests);
            xrequests=requests;
        } catch(...) {
            Error("Could not allocate storage for", "requests");
        }
        char* cp=FileStore();
        for (long i=0; i<numRequests; i++)
        {
            assert(*cp);
            cp=requests[i].Init(cp, i);
        }
        qsort(requests, numRequests, sizeof(Request), Cmp<Request>);
    }
    ~TeachersFile() {delete [] requests;}
};

class OutputFile:public TextFile
// Handles the PT-conferences output file.
{
    FILE* f;
public:
    OutputFile(long testNumber):f(0)
    {
        SetFilename(outputFileName, testNumber);
    }
    void Open()
    {
        f=fopen(outputFileName, "w");
        assert(f);
    }
    void Close() {fclose(f);}
    void SendConference(Conference& C)
    {
        assert(C.IsAssigned());
        fprintf(f, "%s.%s.%d\n",
                C.TheTeacher()->Name(), C.TheParent()->Name(), C.Period());
    }
};

class LogFile
// Handles the PF-log file.
// Log events are stored here, and output together with a single routine.
{
    long numTests;
    double* executionTime;
public:

```

```

    LogFile(long n) :
        numTests(n),
        executionTime(new double[1+n])// tests are numbered 1 to n
    {
        memset(executionTime, 0, sizeof(double)*(1+n));
    }
    ~LogFile() {delete [] executionTime;}
    void Set(long testNumber, double time)
    {
        executionTime[testNumber]=time;
    }
    void Send()
    {
        FILE* f=fopen(logFileName, "w");
        assert(f);
        for (long testNumber=1; testNumber<=numTests; testNumber++)
        {
            fprintf(f, "%10.3f\n", executionTime[testNumber]);
        }
        fclose(f);
    }
};

#endif

```

MYTIMER.H

```

#ifndef MYTIMER_H
#define MYTIMER_H

/* Basic microsecond timer class. */
#include <Timer.h>

class MyTimer
{
    UnsignedWide startTime;
public:
    MyTimer() {Restart();}
    void Restart() {Microseconds(&startTime);}
    double ElapsedMilliSecs() const
    {
        UnsignedWide endTime;
        Microseconds(&endTime);
        return ((endTime.lo-startTime.lo)/1000.0);
    }
};

#endif

```

PT-COMPILE-CONSTANTS.H

```

#ifndef PT_COMPILE_CONSTANTS_H
#define PT_COMPILE_CONSTANTS_H

/*
    Compile policies.
    Mostly self-explanatory.
*/

#define DBG 0

#ifdef __profile__
#undef DBG
#define DBG 0
#endif

#ifdef !DBG
#define NDEBUG
#endif

#include <assert.h>

#define xINLINE_RECURSIVES 1

#define INPUT_DIRECTORY //":InputData:"

#define MAXNUMPERIODS 255

#endif

```


**Are You
Prepared
For An
Emergency?**



www.radgad.com



The Place for Useful Gifts & Gadgets.SM

877-5-RADGAD

MacTech News

The below news headlines recently crossed our news desk. For more information on each of these items, check out the MacTech web site <<http://www.mactech.com>> and search out the below headlines from the home page.

- Berkeley Releases 802.11b WLAN Security Monitoring Device
- VINC v1.0 For Mac OS X and Mac OS
- ChronoSync Updated to Version 1.0.2
- XRay 1.0 - top "Get Info" utility for Mac OS X
- NowPak Module Transfers Contacts to iPod
- Alarm Clock Pro 5.2
- Mergemill 1.5.3 with significant improvements
- IntelliMerge 2.1.5: Adding iPod Support
- Karelia Ships Watson 1.5 and Updated SDK
- ABSplus 60GB now shipping
- StickyBrain is FREE. For two days only.
- Terra Soft Ships Yellow Dog Linux 2.2
- Carbon Events plug-in for REALbasic, version 2.5
- REAL Software announces this year's Cubies contest
- Active4D 2.04 released
- Troll Touch announces TouchStar2 touch screen system for LCD iMac
- FileXaminer 1.0b1 Released
- Dantz Ships Retrospect Backup 5.0 For Mac OS X
- Imagination Software Announces Web Application Development
- SQL4X Manager J released
- Remote Control Atomik 3.0
- Bare Bones Software Releases Mailsmith 1.5 for Mac OS X
- MacHack 17: Keynote Announced: Tim O'Reilly
- Fwd: Press Information: SQL4X Manager J released
- TCP Server Deux v1.0.0 Released by Deep Sky Technologies
- ASG Announces ZipCenter 1.6 Update and Q2 2002 Data
- Revolution 1.1.1 beta 2 (RC1) for Windows, Mac and Linux
- STIMULUS 2.0
- MACVCD X 4.0 RELEASED - Play VCDs on OS X
- Apple In Volume Production of New iMacs, \$100 Price Increase
- Apple Introduces 10GB iPod-2,000 Songs in Your Pocket
- Apple Unveils Cinema HD 23-inch Flat Panel Display
- Apple Previews Bluetooth Wireless Connectivity for Mac OS X
- IPNetMonitor for OS X
- Web Page Calendar Maker 1.0
- NetCloak 3.2 For OS X
- PrefsOverload 4.1.1 For Mac OS And Mac OS X
- IntelliMerge 2.1 For Mac OS
- VVI Introduces Vwidget(TM) Product Name (Formerly called
- illumineX adds new modules to Screen Saver Pack 1.1.2 for Mac OS X
- MacinMind Software updates the server account utility, Passenger
- HEURIS Ships Xtractor
- WiebeTech FireWire DriveDock Shipping
- REALdb Tools 1.2 now available
- New REALbasic database and Microsoft Office Automation plug-ins
- REALbasic 4.0.2 Now Available
- iReplace 1.1 advanced find and replace for html
- Koingo Updates Alarm Clock Pro
- CPLAT 4.0 released
- Iconfactory Releases IconViewer 1.0
- Post.Office Is Alive And Well On Mac OS X
- Captain FTP 1.2 for Mac OS X
- Troi Grabber Plug-in 1.4b1 for FileMaker Pro 5.5
- acgi dispatcher for Mac OS X
- MacsDesign Studio Releases Web Help Desk 5.4
- The Omni Group Announces OmniOutliner 2.0 public beta
- Chronos Group Organizer Server Now In Cocoa
- Niemeijer Announces On-Screen Keyboard Accessibility for Mac OS X
- PC-Mac-Net FileShare transfers files easily between PCs and Macs

**Lucid FireWire
CD-RW!**

**JUST
RELEASED**

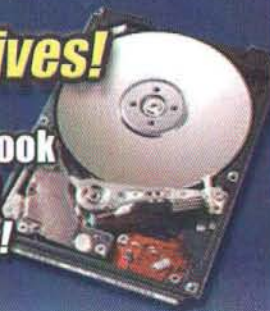


only \$299

**Smallest Drive Available!
Bus Powered!**

Hard Drives!

**For PowerBook
or Desktop!
Best Prices!**



**REALbasic 4.0!
Boxed Edition!**



In Stock Now!

**Includes
Printed
Manuals!**

**REALbasic
Visual QuickStart Guide**

**only
\$19.95**

Pre-Order Now!



**DEV
DEPOT**

**DevDepot your source for the
hottest new Tech Tools!**

www.devdepot.com

Voice: 877-DEPOT-NOW (877-337-6866)
Outside US/Canada: 805/494-9797
E-mail: orders@devdepot.com

List of Advertisers

4D, Inc.	1
AEC Software	41
Aladdin Knowledge Systems, Inc.	35
Aladdin Systems, Inc.	5
Appgen Business Software, Inc.	53
Ascending Technologies	65
Blue World Communications, Inc.	17
CMS Peripherals Inc.	59
Creative Box	71
Crescendo Software	70
DevDepot	28-29
DevDepot.	77
DevDepot.	79
Dr. Bott LLC	37
edition.net	19
Einhugur Programming Resources	70
FairCom Corporation	39
Felt Tip Software	23
Fetch Softworks	13
Flickinger Software	70
Full Spectrum Software, Inc.	27
Inprise / Borland	47
James Sentman Software	72
Jiiva, Inc.	51
Lextek International	27
Mac Show Productions	56
MacTech Magazine	31
Management Software Inc.	7
Marathon Computers, Inc.	61
Mathemaesthetics, Inc.	25
Metrowerks Corporation, A Motorola Company	BC
Montage Technologies, Inc.	73
MYOB US, Inc.	67
Netopia, Inc.	63
O'Reilly & Associates, Inc.	54
Onyx Technology	19
OpenBase International, Ltd.	45
PagePlanet Software, Inc.	49
Paradigma Software	60
Peachpit Press	78
piDog Software	71
Power On Software	33
PrimeBase (SNAP Innovation)	21
R/com Networks, Inc.	55, 57
RadGad	75
REAL Software, Inc.	9, 69
RiverSong InterActive	72
Runtime Revolution Limited	IFC
Scientific Placement, Inc.	23
StimpSoft, Inc.	72
SuSE Inc.	IBC
Thursby Software Systems, Inc.	11
WIBU-SYSTEMS AG	43
Xochi Media Inc.	71
Zero G Software	15

List of Products

4D Web Edition * 4D, Inc.	1
Accessories * DevDepot	28-29
Accounting Products * Appgen Business Software, Inc.	53
Adobe Press Ad * Peachpit Press	78
Application Builder Collection * Jiiva, Inc.	51
Backup Systems & Peripherals * CMS Peripherals Inc	59
Books * O'Reilly & Associates, Inc.	54
c-tree Plus * FairCom Corporation	39
CalendarMonster * Ascending Technologies	65
Career Opportunities * Scientific Placement, Inc.	23
CodeWarrior * Metrowerks Corporation, A Motorola Company	BC
Corona * Flickinger Software	70
DAVE * Thursby Software Systems, Inc.	11
DevDepot News Specials * DevDepot	79
Development Services * Montage Technologies, Inc.	73
Development & Testing * Full Spectrum Software, Inc.	27
DVIator * Dr. Bott LLC	37
FastTrack * AEC Software	41
Fetch * Fetch Softworks	13
FontViewer * Creative Box	71
HASP4 for Mac * Aladdin Knowledge Systems, Inc.	35
InstallAnywhere * Zero G Software	15
InstallerMaker * Aladdin Systems, Inc.	5
iScreensaver Designer * Xochi Media Inc.	71
JBUILDER * Inprise / Borland	47
JobOrder * Management Software Inc.	7
Lasso * Blue World Communications, Inc.	17
Mac Show Live * Mac Show Productions	56
MacTech CD * MacTech Magazine	31
MGI * PagePlanet Software, Inc.	49
MYOB * MYOB US, Inc.	67
Now Up-to-Date & Contact * Power On Software	33
Onix, RouteX, Brevity * Lextek International	27
OpenBase * OpenBase International, Ltd.	45
Picture Play * Crescendo Software	70
piDock * piDog Software	71
PrimeBase * PrimeBase (SNAP Innovation)	21
R/Com TV * R/com Networks, Inc.	55, 57
Rackmount Solutions * Marathon Computers, Inc.	61
RAM * DevDepot	77
REALbasic Plug-ins * Einhugur Programming Resources	70
REALbasic * REAL Software, Inc.	9
Reseller Hosting * edition.net	19
Resorcerer * Mathemaesthetics, Inc.	25
Revolution * Runtime Revolution Limited	IFC
Software for Cats * StimpSoft, Inc.	72
Software Protection * WIBU-SYSTEMS AG	43
Sound Studio * Felt Tip Software	23
Spotlight * Onyx Technology	19
SuSE Linux * SuSE Inc.	IBC
Timbuktu Pro & netOctopus * Netopia, Inc.	63
TitleTrack Jukebox * RiverSong InterActive	72
Useful Gifts & Gadgets * RadGad	75
Valentina * Paradigma Software	60
Whistle Blower * James Sentman Software	72

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

- **SECURE**
- **STABLE**
- **SUPERIOR**

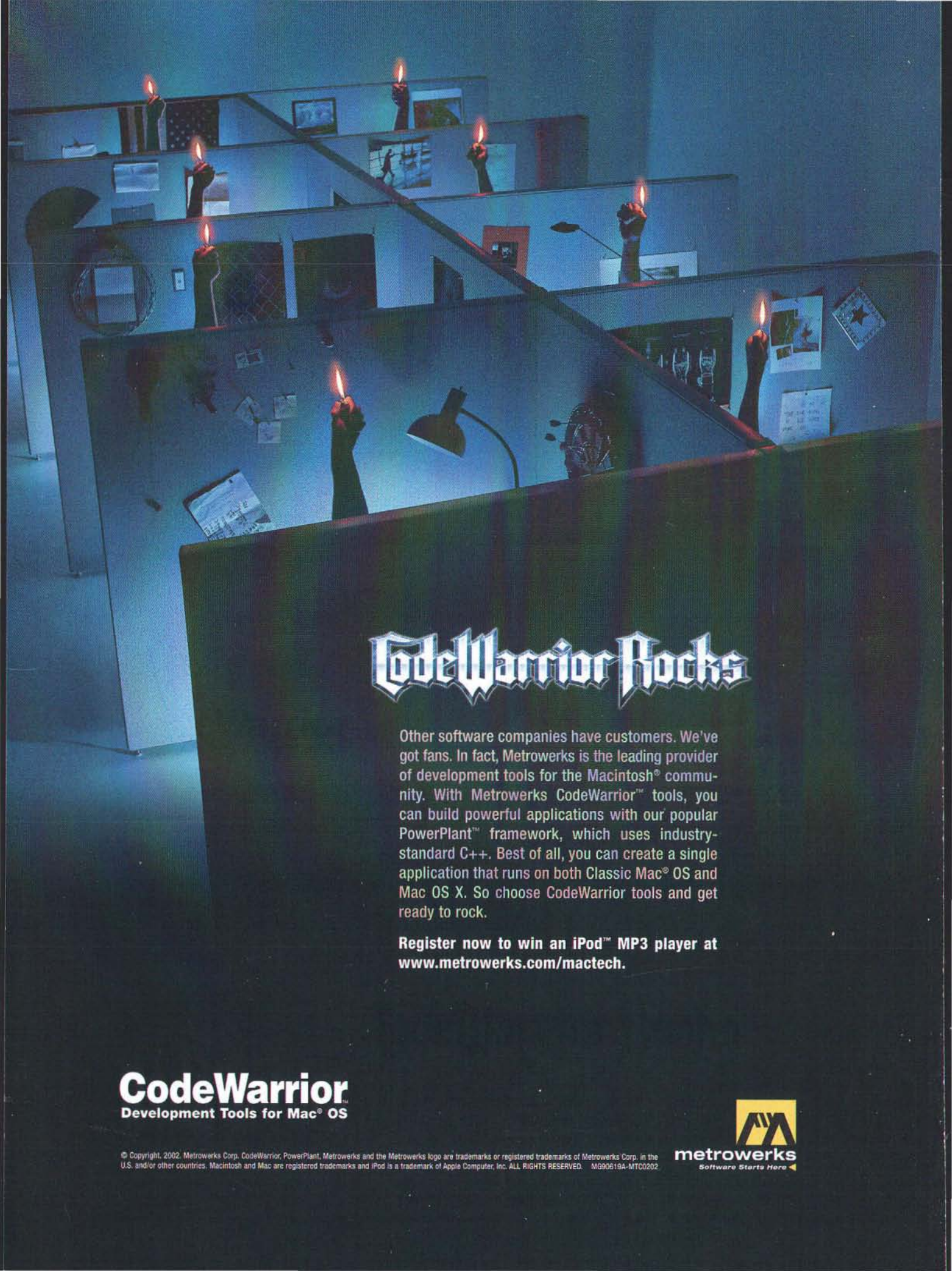


... because Linux users enjoy the power and stability of SuSE Linux.
Don't compromise.

- Full integration with MOL (Mac-On-Linux)
- Scanning made easy with the KDE tool Kooka **NEW**
- Protect your data with Soft-RAID **NEW**
- Great sound with ALSA
- Surf safely with the Personal Firewall
- Professional installation support by phone, fax and e-mail
- etc.

☐ SuSE Linux 7.3 PowerPC EDITION
\$79.95





CodeWarrior Rocks

Other software companies have customers. We've got fans. In fact, Metrowerks is the leading provider of development tools for the Macintosh® community. With Metrowerks CodeWarrior™ tools, you can build powerful applications with our popular PowerPlant™ framework, which uses industry-standard C++. Best of all, you can create a single application that runs on both Classic Mac® OS and Mac OS X. So choose CodeWarrior tools and get ready to rock.

Register now to win an iPod™ MP3 player at www.metrowerks.com/mactech.

CodeWarrior
Development Tools for Mac® OS

© Copyright, 2002, Metrowerks Corp. CodeWarrior, PowerPlant, Metrowerks and the Metrowerks logo are trademarks or registered trademarks of Metrowerks Corp. in the U.S. and/or other countries. Macintosh and Mac are registered trademarks and iPod is a trademark of Apple Computer, Inc. ALL RIGHTS RESERVED. MG90619A-MTC0202

